CQNCR: Optimal VM Migration Planning in Cloud Data Centers

Md. Faizul Bari, Mohamed Faten Zhani, Qi Zhang, Reaz Ahmed, and Raouf Boutaba
David R. Cheriton School of Computer Science
University of Waterloo, Ontario, Canada
[mfbari | mfzhani | q8zhang | r5ahmed | rboutaba]@uwaterloo.ca

***Abstract*—With the proliferation of cloud computing, virtualization has become the cornerstone of modern data centers and an effective solution to reduce operational costs, maximize utilization and improve performance and reliability. One of the powerful features provided by virtualization is Virtual Machine (VM) migration, which facilitates moving workloads within the infrastructure to reach various performance objectives. As recent virtual resource management schemes are more reliant on this feature, a large number of VM migrations may be triggered simultaneously to optimize resource allocations. In this context, a challenging problem is to find an efficient migration plan, i.e., an optimal sequence in which migrations should be triggered in order to minimize the total migration time and impact on services.**

In this paper, we propose CQNCR (read as sequencer), an effective technique for determining the execution order of massive VM migrations within data centers. Specifically, given an initial and a target resource configuration, CQNCR sequences VM migrations so as to efficiently reach the final configuration with minimal time and impact on performance. Experiments show that CQNCR can significantly reduce total migration time by up to 35% and service downtime by up to 60%.

I. Introduction

With the rise of cloud computing, virtualization has become a key technology that enables IT companies to better manage their data centers and cloud environments. Recent advances in virtualization have allowed to partition a cloud data center into multiple Virtual Data Centers (VDCs) intended to host and run different applications and services in a completely isolated manner [4], [5], [10], [21]. Specifically, a VDC encompasses Virtual Machines (VMs) and virtual links with guaranteed bandwidth capacity, allowing to provide predictable computing and networking performance.

Of course, several management frameworks have been proposed to efficiently manage VDCs and achieve various objectives related to performance, cost and survivability. In this context, VM migration has been the ultimate tool to dynamically reconfigure and re-optimize the resource allocations [6], [12], [18], [21]. Depending on the targeted objective, a set of VM migrations can be triggered to reach the required resource configuration. However, these migrations may have several impacts on network and service performance:

- *Resource consumption*. The migration process consumes resources (e.g., CPU, memory) either at the source or at the destination physical machines [18], [19]. This significantly impacts the performance of other VMs sharing these machines [16]. Indeed, although theoretically VMs have dedicated and isolated resources, in practice, current hypervisors are not able to guarantee effective performance isolation [19].
- *Network congestion.* When massive migrations are triggered, VMs may be transferred using migration paths sharing some common links. This may eventually cause network congestion, resulting in degraded performance in terms of latency and packet loss.
- *Long provisioning time.* A long duration of the migrations may prevent cloud providers from allocating new VDCs or may prolong the resource allocation process. For instance, resource allocation scheme proposed in [12], [21] requires to migrate some VMs in order to make room for any incoming VDC. In this case, it is necessary to reduce their migration time in order to allocate resources for the new VDC with the least possible delay.
- *Service disruption time.* Several techniques have been proposed in the literature to ensure a seamless and efficient live migration [6]. Nevertheless, VMs still experience non-negligible downtimes during their migrations [15], [16]. In practice, any service disruption, even for very short periods, may have serious impact on service revenues and cloud provider's reputation. For some services, one second of downtime may result in as much as $\$1,100$ losses in revenue [1].

To alleviate these impacts, one possible solution is to minimize the total migration time of the VMs by finding an efficient VM migration plan i.e., an optimal sequence of migrations ensuring that (1) the amount of bandwidth used for migration is maximized, (2) the migration paths are carefully selected to avoid network congestion, and (3) the number of parallel migrations of VMs is maximized, notably of those belonging to the same VDC so as to reduce its disruption time.

As network resources are allocated for the VDCs, VMs are migrated using only the residual (i.e., unused) bandwidth of the physical links.[1] Hence, the higher is the residual bandwidth, the lower is the migration time. Typically, when a VM is moved to a new location, its associated virtual links are also migrated. This changes the residual bandwidth in the

[1]We assume that there is no alternate network dedicated to VM migrations since its deployment is costly and may not be possible in large-scale infrastructures [16]. Nevertheless, even if such network is set up, our solution can still be used to achieve objectives (2) and (3) mentioned above.

physical links, which, in turn, impacts the duration of the subsequent migrations and their incurred service disruption time; Hence the need to find an optimal migration plan that minimizes the total migration time and impact on the network and service quality.

To address this challenge, we propose CQNCR (read as sequencer), a migration manager that orchestrates and executes massive VM migrations in data centers with the goal of minimizing their overall migration time and overhead. Unlike previous work [3], [11], [12], the proposed solution takes into consideration the change of residual bandwidth at each step of the migration plan and also maximizes parallel migrations while avoiding potential bottlenecks in the network. More importantly, CQNCR can be easily incorporated into VM/VDC management frameworks like SandPiper [18], VDC Planner [21] and SecondNet [10] to further optimize the execution of the triggered migrations and improve the overall performance of the network and services.

The rest of the paper is organized as follows. We first provide background material in Section II. Section III presents an illustrative example showing how different migration plans can impact the total migration and service disruption times. Section IV surveys the related work on VM/VDC management frameworks and migration planning. We then formulate the VDC migration planning problem as an Integer Linear Program (ILP) in Section V. The proposed solution is presented in Section VI. Then we provide performance evaluation results in Section VII and some conclusions in Section VIII.

II. Background

A large body of work has been devoted to live migration techniques [6]. The main goal of live migration is to relocate a VM with the minimum possible service disruption time. Generally speaking, two main approaches were proposed in the literature: post-copy and pre-copy migration [13].

As the pre-copy migration is the most common approach that is already implemented in the state-of-the-art hypervisors (e.g., VMware, XEN, KVM) [13], we adopt this approach in this work, and hence we only provide its details in the following paragraph. As shown in Fig. 1, in a pre-copy live migration, the VM content (i.e., VM image) is first copied from the source physical machine to the destination host (i.e., *VM image copy* phase). Subsequently, the *iterative pre-copy* phase is started. During the first iteration of this phase, all memory pages are copied to the new VM. As memory content can change during the transfer, the subsequent iterations will only copy the changed memory pages (called dirty pages) to the target machine. The dirty pages are continuously copied until their number becomes relatively small, or a certain number of iterations is reached. This marks the start of the next phase, called the *stop-and-copy* phase, where the source VM is suspended, and the dirty pages are copied to the destination. Then, the new VM at the destination host takes over while the source VM is destroyed. It is worth noting that the VM is not available only during the stop-and-copy phase.

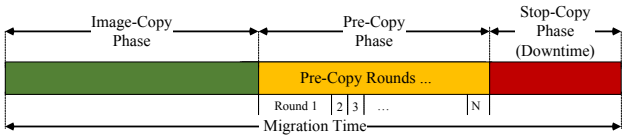


Fig. 1. Pre-copy Live Migration [13]

Mann *et al.* provide a mathematical model for live migration in VMware vMotion and KVM [15], [16]. They compute the total migration time using the following parameters: the VM image size denoted by W (MB), the VM memory size M (MB), the VM page dirty rate R (MBps), and the amount of bandwidth used for migrating the VM denoted by L MBps, the expected duration for the stop-copy phase denoted by T (seconds), and amount of memory transferred in each pre-copy round denoted by X (MB). The duration of the image-copy phase can be written as:

$$T_i = \frac{W}{L} \tag{1}$$

The duration of the pre-copy and stop-copy phase is

$$T_{p+s} = \frac{M.\frac{1-(R/L)^n}{1-(R/L)}}{L} \tag{2}$$

Finally, the duration of only the stop-copy phase is

$$T_s = \frac{M}{L}.\left(\frac{R}{L}\right)^n \tag{3}$$

where

$$n = \min\left(\left\lceil \log_{R/L}\frac{T.L}{M}\right\rceil, \left\lceil \log_{R/L}\frac{X.R}{M.(L-R)}\right\rceil\right) \tag{4}$$

The total migration time can be given by the sum of the durations of theses phases (*i.e.*, $T_i + T_{p+s}$). The downtime for a VM is given by T_s.

III. Motivation

In this section, we provide a motivating example to show the impact of migration sequence planning on the total migration time, VDC downtime, and VM downtime.

In this example, we consider a network consisting of four servers (P1, P2, P3, and P4), two top-of-rack switches, and two aggregation switches (Fig. 2(a)). Each server has 16 cores, 32 GB of memory, and 1 TB of disk and is connected to a top-of-rack switch with a 1-Gbps link. Switches are inter-connected via 10-Gbps links. The resulting topology is a typical tree topology as shown in Fig. 2(a).

We assume two VDCs are embedded in the considered data center (Fig. 3(a) and Fig. 3(b)). VDC 1 contains five VMs and six virtual links, whereas VDC 2 contains five VMs and four virtual links. Fig. 3(a) and Fig. 3(b) respectively show the virtual topologies of VDC 1 and VDC 2, as well as resource requirements of each of their VMs (*i.e.*, CPU, memory, and image size) and virtual links (*i.e.*, bandwidth).

Fig. 2(a) and Fig. 2(b) show two possible resource mappings of these VDCs onto the physical topology. For instance,

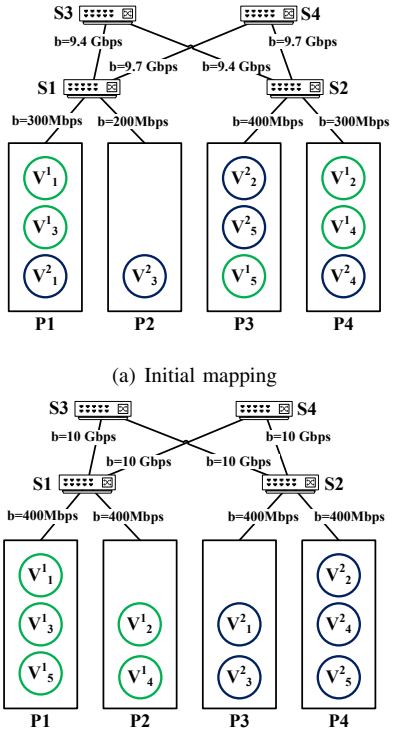
Fig. 2(a) shows the following mapping for VDC 1: the VMs V_1^1 and V_3^1 are embedded in P1, V_2^1 and V_4^1 are embedded in P4 and V_5^1 in P3. The virtual links (V_1^1, V_2^1), (V_3^1, V_4^1), (V_2^1, V_5^1) and (V_3^1, V_5^1) are mapped onto the paths (P1, S1, S4, S2, P4), (P1, S1, S4, S2, P4), (P4, S2, P3) and (P1, S1, S4, S2, P3), repectively. Finally, the virtual links (V_1^1, V_3^1) and (V_2^1, V_4^1) are embedded within the physical machines P1 and P4, respectively. The figures also show the residual bandwidth (denoted by b) for each physical link *i.e.*, the bandwidth available after embedding the virtual links.

Our objective is to devise a sequence of migrations to move VMs from the initial mapping (Fig. 2(a)) to the final one (Fig. 2(b)) with minimal migration time, VM and VDC downtime.

Two possible migration sequences are shown in Fig. 4(a) and Fig. 4(b). In each figure, the bottom line represents time in seconds and the top bounded lines represent start, duration, and end of a VM migration. The name of the VMs are shown on the left side. For instance, in the first sequence (Fig. 4(a)), two migrations are triggered at time 0: (i) VM V_1^2 is migrated from P1 to P3, and (ii) VM V_5^2 is migrated from P3 to P4. The first migration takes 10 seconds and the second one takes 60 seconds. When VM V_1^2 is migrated from P1 to P3, its associated virtual links are removed from P1 and re-embedded at P3. Hence, the residual bandwidth in links changes after each step in the sequence. Subsequently, V_2^2 is migrated from P3 to P1 using the available residual bandwidth. Migrations are executed according to the provided sequence until the final mapping is reached.

For the first sequence in Fig. 4(a), the total migration time is 220 seconds, and for the second one in Fig. 4(b), the total migration time is 165 seconds. The average VM downtime (*i.e.*, average duration of the stop-copy phase) for the two sequences are 4.14 and 3.9 seconds, respectively. The average VDC downtime for these two sequences are 14.5 and 13.7 seconds, respectively. Here, a VDC is assumed to be down if one of its VMs is down.

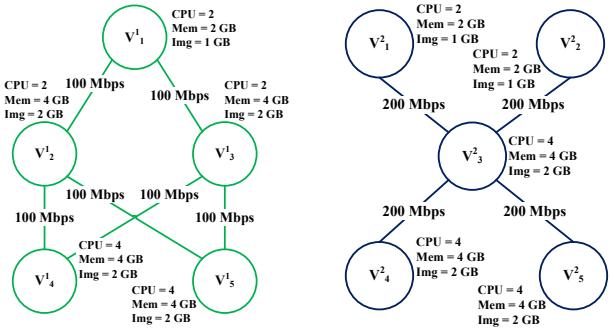
While we perform parallel migrations in both sequences, the second sequence provides better performance as it schedules the migrations in a way that maximize the residual bandwidth for the subsequent migrations. For this simple example with two VDCs and a small-scale data center, the second sequence provides more than 25% improvement in total migration time over the first one. As we shall show in Section VII, for larger-scale data centers, the migration plan can have higher impact on the performance, and our proposed solution can further reduce total migration time by up to 35% and VM/VDC downtime by up to 60%.



(a) Initial mapping

(b) Final mapping

Fig. 2. Initial and final mappings



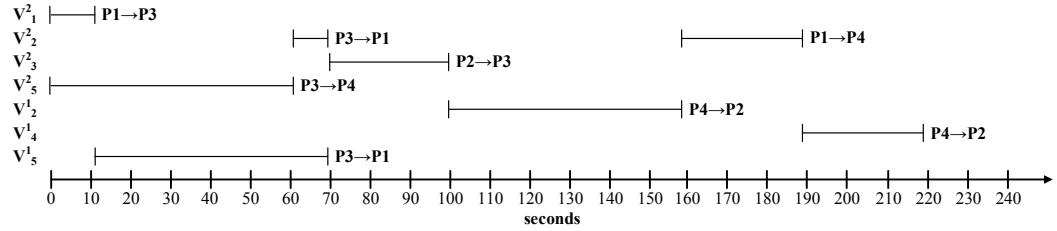
(a) VDC 1 (b) VDC 2

Fig. 3. VDC Configurations

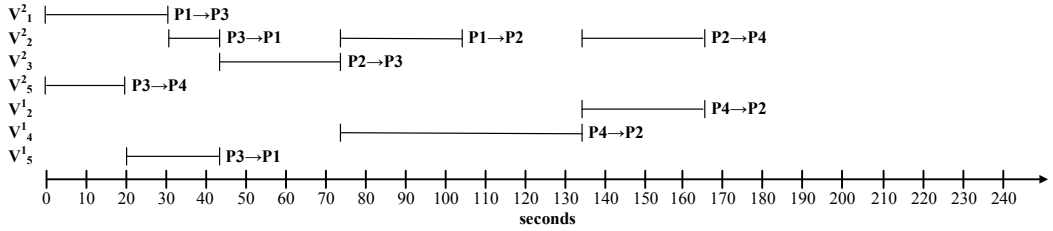
IV. Related Work

VM Migration has been a central tool allowing dynamic resource management and optimization in cloud environments. Recently, several proposals have relied on VM migration to manage VMs and VDCs in virtualized data centers [12], [18], [21]. For instance, Sandpiper is a VM management scheme for data center environments designed to avoid machine overload by relocating VMs [18]. Another work that also relies on VM migration is VDC Planner [21]. VDC Planner is a VDC management framework that leverages VM migration to handle various usage scenarios, namely VDC embedding and scaling up and down, and also server consolidation. In this framework, migration is used to re-optimize the current VM mapping in order to make room for new VDCs, or to allow adding more VMs and virtual links to a previously embedded VDC. However, for both frameworks, Sandpiper and VDC Planner, the impact of migrations' order was not studied and VMs are moved without a prior setup of a particular plan.

The first work that considered finding optimal migration plans is that of Hermenier *et al.* [11], [12]. They proposed



(a) Sequence 1



(b) Sequence 2

Fig. 4. VM migration sequences

Entropy, a resource manager that dynamically consolidate workloads in data centers. Through experiments, they found that the downtime of a migrated VM (which corresponds to the duration of stop-and-copy phase [7], [12]) depends mainly on the size of its memory. Based on this observation, they developed a migration cost model that considers only the VM memory size. Then they formulated the migration sequencing problem as a Constraint Satisfaction Problem (CSP) that aims at minimizing the migration cost. The solution that we propose in this paper is different from Entropy as we aim at minimizing the total migration time and also the downtime of a whole VDC. Al-Haj *et al.* addressed also the VM migration planning problem. However, their main focus was to ensure the satisfaction of security, capacity, and performance constraints during the execution of the migration plan [3]. Ghorbani *et al.* proposed a heuristic algorithm to migrate virtual machines in OpenFlow-enabled networks [9], [17]. However, their algorithm does not aim at minimizing the migration time or service disruption time but rather, it aims at finding a migration plan that does not introduce any inconsistent state in the network. Lo *et al.* also tackled the same problem but in the context of WAN virtualization [14]. They proposed three algorithms for migrating multiple virtual routers with the goal of minimizing migration time and cost. Kikuchi *et al.* investigated the performance and overhead related to live migration strategies for virtual clusters [20]. They highlighted the scarcity of network bandwidth and stressed the need to determine optimal migration orders in order to efficiently use bandwidth. However, they do not provide a solution to this problem in their work.

Unlike the aforementioned studies, our solution finds the optimal migration plan while considering the change in residual bandwidth and the inter-dependence between VMs. Indeed, when offering VDCs instead of simple VMs, the migration of a VM involves also the migration of its associated virtual links. Thus, the residual bandwidth changes at each step of the migration plan, which impacts the total migration time and service disruption time. Furthermore, our solution tries to migrate simultaneously VMs belonging to the same VDC in order to minimize the downtime of the service offered by the VDCs.

V. The VDC Migration Sequencing Problem

In this section we formally define the VDC migration sequencing problem as an Integer Linear Program (ILP). In our model, the physical data center network is represented by a graph $\bar{G} = (\bar{N}, \bar{L})$, where $\bar{N}$ and $\bar{L}$ denote the nodes and links respectively in the data center network. Furthermore, we define $\bar{N} = \bar{N_H} \cup \bar{N_S}$, where $\bar{N_H}$ and $\bar{N_S}$ represent the set of physical hosts and switches, respectively. The CPU, memory and disk capacities of each physical host $\bar{n} \in \bar{N_H}$ are denoted by $\bar{\mathcal{C}}_{\bar{n}} \in \mathbb{R}^+$, $\bar{\mathcal{M}}_{\bar{n}} \in \mathbb{R}^+$ and $\bar{\mathcal{D}}_{\bar{n}} \in \mathbb{R}^+$, respectively. Bandwidth capacity of each physical link $\bar{l} \in \bar{L}$ is represented by $\bar{\mathcal{B}}_{\bar{l}} \in \mathbb{R}^+$. We also define $s_{\bar{n}\bar{l}} \in \{0,1\}$ and $d_{\bar{n}\bar{l}} \in \{0,1\}$ to indicate whether $\bar{n}$ is the source and destination of each physical link $\bar{l}$:

$$s_{\bar{n}\bar{l}} = \begin{cases} 1 & \text{if } \bar{n} \in \bar{N} \text{ is the source of } \bar{l} \in \bar{L}, \\ 0 & \text{otherwise.} \end{cases}$$

$$d_{\bar{n}\bar{l}} = \begin{cases} 1 & \text{if } \bar{n} \in \bar{N} \text{ is the destination of } \bar{l} \in \bar{L}, \\ 0 & \text{otherwise.} \end{cases}$$

Let I denote the set of VDCs embedded in the data center. Each VDC $i \in I$ is represented as a graph $G^i = (N^i, L^i)$, where N^i is the set of VMs and L^i is the set of virtual links between VMs in N^i. For each VM $n \in N^i$, let $\mathcal{C}^i_n \in \mathbb{R}^+$, $\mathcal{M}^i_n \in \mathbb{R}^+$ and $\mathcal{D}^i_n \in \mathbb{R}^+$ represent the CPU, memory and disk

(*i.e.*, image size) requirements, respectively. $\mathcal{R}_n^i$ represents the average page dirty rate for n. For each virtual link $l \in L^i$, let $\mathcal{B}_l^i \in \mathbb{R}^+$ denote the bandwidth requirement of virtual link l and $\zeta_{l\bar{l}}^i \in \mathbb{R}^+$ represent the amount of physical link $\bar{l} \in \bar{L}$ bandwidth allocated to virtual link $l \in L^i$. Furthermore, the source and destination of each virtual link l is specified by the following variables:

$$s_{nl}^i = \begin{cases} 1 & \text{if } n \in N^i \text{ is the source of } l \in L^i, \\ 0 & \text{otherwise.} \end{cases}$$

$$d_{nl}^i = \begin{cases} 1 & \text{if } n \in N^i \text{ is the destination of } l \in L^i, \\ 0 & \text{otherwise.} \end{cases}$$

The mapping between the VMs in VDC i and the physical hosts in $\bar{N}_H$ at time t is represented by the following binary variable:

$$y_{n\bar{n}}^{it} = \begin{cases} 1 & \text{if } n \in N^i \text{ is embedded in } \bar{n} \in \bar{N_H} \text{ at time } t, \\ 0 & \text{otherwise.} \end{cases}$$

To capture the system state during the migration sequence, we adopt a discrete time model where time is divided into intervals of fixed length. A new migration can start at the beginning of a time interval. Even though in practice the VM migration may happen at any given time, we choose to adopt the discrete time model mainly due to its simplicity. Furthermore, this discrete time model can be arbitrarily close to the continuous time model as the interval length becomes infinitely small. Let $[0, T]$ denote the time interval in which the migrations take place, where T is maximum time allowed for the migration sequence. The mapping of all embedded VDCs at a particular time t is then captured by the multi-dimensional matrix $(y_{n\bar{n}}^{it})_{i\in I, n\in N_i, \bar{n}\in\bar{N}}$.

In the VDC migration sequencing problem, the initial and final placement of VMs, (*i.e.*, $(y_{n\bar{n}}^{i0})_{i,n,\bar{n}}$ and $(y_{n\bar{n}}^{iT})_{i,n,\bar{n}}$ at time 0 and T, respectively), are provided as inputs. Our goal is to find a sequence of states $(y_{n\bar{n}}^{it})_{i,n,\bar{n}}$, $0 < t < T$ such that system gradually transits from the initial state $(y_{n\bar{n}}^{i0})_{i,n,\bar{n}}$ to $(y_{n\bar{n}}^{iT})_{i,n,\bar{n}}$:

$$(y_{n\bar{n}}^{i0})_{i,n,\bar{n}} \to (y_{n\bar{n}}^{i1})_{i,n,\bar{n}} \to (y_{n\bar{n}}^{i2})_{i,n,\bar{n}} \to \ldots \to (y_{n\bar{n}}^{iT})_{i,n,\bar{n}}.$$

This is achieved by scheduling migrations at appropriate times. Define $z_{n\bar{p}\bar{q}}^{it} \in \{0, 1\}$ as a variable that indicates whether n is to be migrated from $\bar{p}$ to $\bar{q}$ at t:

$$z_{n\bar{p}\bar{q}}^{it} = \begin{cases} 1 & \text{if } n \in N^i \text{ is scheduled to migrate from } \bar{p} \text{ to } \bar{q} \\ & \text{at time } t \\ 0 & \text{otherwise.} \end{cases}$$

As a migration may take several periods to complete, to account for migration time, we define $X_{n\bar{p}\bar{q}}^{it} \in \{0, 1\}$ as a variable that indicates whether n is under migration during time period t:

$$X_{n\bar{p}\bar{q}}^{it} = \begin{cases} 1 & \text{if } n \text{ is migrating from } \bar{p} \text{ to } \bar{q} \text{ during period } t \\ 0 & \text{otherwise.} \end{cases}$$

To ensure that $z_{n\bar{p}\bar{q}}^{it}$ triggers the migration, and only a single migration can take place for a single VM n, we must have the following constraints:

$$X_{n\bar{p}\bar{q}}^{it} \geq z_{n\bar{p}\bar{q}}^{it}, \quad \forall i \in I, n \in N^i, \bar{p}, \bar{q} \in \bar{N}_H, 0 \leq t \leq T \quad (5)$$

$$\sum_{\bar{q}\in N} X_{n\bar{p}\bar{q}}^{it} \leq 1, \quad \forall i \in I, n \in N^i, \bar{p}, \bar{q} \in \bar{N}_H, 0 \leq t \leq T \quad (6)$$

Let $b_{n\bar{p}\bar{q}}^{it} \in \mathbb{R}^+$ represent the bandwidth allocated for migration of n from $\bar{p}$ to $\bar{q}$ at time t, and let $r_{n\bar{p}\bar{q}}^{it} \in R^+$ denote the size of the remaining content to be copied from $\bar{p}$ to $\bar{q}$ for VM n at the beginning of time t. The following constraint states that when migration starts, the total content to be copied is $\mathcal{M}_n^i$ (*e.g.*, the memory and image size of the VM):

$$r_{n\bar{p}\bar{q}}^{it} = z_{n\bar{p}\bar{q}}^{it} \mathcal{M}_n^i, \forall i \in I, n \in N^i, \bar{p}, \bar{q} \in \bar{N}_H, 0 \leq t \leq T \quad (7)$$

Furthermore, unless $r_{n\bar{p}\bar{q}}^{it}$ becomes negative, when time period $t+1$ starts, $r_{n\bar{p}\bar{q}}^{it}$ should be reduced by $b_{n\bar{p}\bar{q}}^{it}$:

$$r_{n\bar{p}\bar{q}}^{i(t+1)} \geq r_{n\bar{p}\bar{q}}^{it} - b_{n\bar{p}\bar{q}}^{it} X_{n\bar{p}\bar{q}}^{it}, \quad \forall i \in I, n \in N^i, \bar{p}, \bar{q} \in \bar{N}_H, 0 \leq t \leq T-1 \quad (8)$$

We also want to ensure that migration must continue (*i.e.*, $X_{n\bar{p}\bar{q}}^{i(t+1)} = 1$) at time $t+1$ if the remaining content after time t is positive:

$$X_{n\bar{p}\bar{q}}^{i(t+1)} \geq (r_{n\bar{p}\bar{q}}^{it} - b_{n\bar{p}\bar{q}}^{it} X_{n\bar{p}\bar{q}}^{it})/\mathcal{M}_n^i, \quad \forall i \in I, n \in N^i, \bar{p}, \bar{q} \in \bar{N}_H, 0 \leq t \leq T-1 \quad (9)$$

When migration of VM n is being carried out, VM n must be present in both the source and destination machines $\bar{p}$ and $\bar{q}$:

$$y_{n\bar{p}}^{it} \geq X_{n\bar{p}\bar{q}}^{it} \quad \forall i \in I, n \in N^i, \bar{p}, \bar{q} \in \bar{N}_H, 0 \leq t \leq T \quad (10)$$

$$y_{n\bar{q}}^{it} \geq X_{n\bar{p}\bar{q}}^{it} \quad \forall i \in I, n \in N^i, \bar{p}, \bar{q} \in \bar{N}_H, 0 \leq t \leq T \quad (11)$$

Furthermore, the total resource used by the VMs running in physical host $\bar{n}$ must satisfy its capacity constraints for both CPU, memory and disk:

$$\sum_{i=1}^{I} \sum_{n\in N^i} y_{n\bar{n}}^{it} \mathcal{C}_n^i \leq \bar{\mathcal{C}}_{\bar{n}}, \qquad \forall t, \bar{n} \in \bar{N}_H \quad (12)$$

$$\sum_{i=1}^{I} \sum_{n\in N^i} y_{n\bar{n}}^{it} \mathcal{M}_n^i \leq \bar{\mathcal{M}}_{\bar{n}}, \qquad \forall t, \bar{n} \in \bar{N}_H \quad (13)$$

$$\sum_{i=1}^{I} \sum_{n\in N^i} y_{n\bar{n}}^{it} \mathcal{D}_n^i \leq \bar{\mathcal{D}}_{\bar{n}}, \qquad \forall t, \bar{n} \in \bar{N}_H \quad (14)$$

In addition, let $v_{\bar{p}\bar{q}\bar{l}} \in \{0, 1\}$ indicates whether the migration path from $\bar{p}$ to $\bar{q}$ includes $\bar{l}$. The following constraint states that the total bandwidth used by VDCs for migration must respect the link capacity constraint of each link $\bar{l} \in \bar{L}$:

$$\sum_{i=1}^{I} \sum_{n\in N^i} \xi_{l\bar{l}}^i + \sum_{\bar{p}} \sum_{\bar{q}} b_{n\bar{p}\bar{q}}^{it} v_{\bar{p}\bar{q}\bar{l}} \leq \bar{\mathcal{B}}_{\bar{l}} s_{\bar{n}\bar{l}}, \forall t, \bar{n} \in \bar{N}_H, \bar{l} \in \bar{L} \quad (15)$$

We also require the flow constraint which states that the total incoming bandwidth should be equal to the total outgoing bandwidth for every node $\bar{n}$ along a virtual link l, unless $\bar{n}$ is either the source node (in which case the net outgoing bandwidth is positive) or the destination node (in which case the net outgoing bandwidth is negative) of l:

$$\sum_{\bar{l}\in\bar{L}} \bar{s}^{it}_{\bar{n}\bar{l}}\zeta^{it}_{\bar{l}\bar{l}} - \sum_{\bar{l}\in\bar{L}} \bar{d}^{it}_{\bar{n}\bar{l}}\zeta^{it}_{\bar{l}\bar{l}} = \sum_{n\in N^i} y^{it}_{n\bar{n}} s^i_{nl}\mathcal{B}_l - \sum_{n\in N^i} y^{it}_{n\bar{n}} d^i_{nl}\mathcal{B}_l \quad \forall i \in I, t, \bar{n} \in N, l \in L^i \tag{16}$$

We must also ensure that at any given moment, every VM must be embedded in at least one physical machine:

$$y^{it}_{n\bar{q}} \geq 1 \qquad \forall i \in I, n \in N^i, \bar{q} \in \bar{N}_H, 0 \leq t \leq T \tag{17}$$

To capture our objective of minimizing total migration time, let $w^t \in \{0,1\}$ represent whether migration occurs at time t. To ensure that migration sequence will be carried out continuously without idle periods, the following constraints must be satisfied:

$$w^t \geq X^{it}_{n\bar{p}\bar{q}} \quad \forall i \in I, n \in N^i, \bar{p}, \bar{q} \in \bar{N}, 0 \leq t \leq T \tag{18}$$

$$w^t \geq w^{t+1} \quad \forall 0 \leq t \leq T-1 \tag{19}$$

Finally, let $P^i_n \in \mathbb{R}^+$ represent the penalty for migrating a VM n in single time period. Thus, the overall objective is to minimize the total migration time and service penalty due to migration. Let $\delta \in \mathbb{R}^+$ denote the weight factor for normalizing the two objectives, the goal of the VDC migration sequencing problem can be stated as follows:

$$\min \left(\sum_{t=0}^{T} w^t + \delta \sum_{i=1}^{I} \sum_{n\in N^i} X^{it}_n P^i_n \right) \tag{20}$$

subject to constraints (5)-(19). This problem is $\mathcal{NP}$-hard to solve because even without CPU and memory capacity constraints, the problem generalizes the data migration problem [8], which is known to be $\mathcal{NP}$-hard.

VI. Sequencer (CQNCR)

In this section, we present CQNCR, our heuristic solution to the VM migration sequencing problem. Given an initial and a target resource allocation, the goal of this algorithm is to find a sequence of VM moves that translates the initial VM embedding to the target one. Before going into the details of the proposed heuristic, we first present various design alternatives that can be considered, and also the rationale behind our design choices.

A. Design Rationale

The goal of the proposed solution is to find a migration sequence that minimizes the total migration and service disruption times as captured by the objective function (Eq. 20). To achieve this goal, the following sights can be used to guide the algorithm to determine an efficient migration plan:

- Start by migrating VMs whose new placement will result in the increase of the residual bandwidth, and thereby ensure that subsequent migrations are performed faster, which reduces also VM downtime.
- Migrate first VMs incurring the least migration time.
- Avoid concurrently transferring VMs to and from the same physical machine in order to reduce the processing overhead introduced by migration. In other words, one physical machine is either receiving or sending one and only one VM at a particular point of time.
- When concurrently transferring VMs, avoid using paths sharing the same links in order to inhibit network congestion due to migration.

Based on these guidelines, we designed our heuristic solution as detailed in the following subsection.

B. Heuristic Solution

Our heuristic solution is described in Algorithm 1. Given the initial ($y^0_{n\bar{n}}$) and target ($y^T_{n\bar{n}}$) mapping, Algorithm 1 returns a set of tuples S_q containing the migration start time, VM to migrate, source and destination physical hosts, and the migration path.

The main idea of the algorithm is to divide VMs to be migrated into Resource Independent Groups (RIGs). Each RIG contains a set of VMs that can be migrated through disjoint paths and between distinct machines. Then VMs of each RIG are migrated simultaneously. RIG migrations are triggered sequentially starting by the RIGs that have shorter migration time and willing to free up more residual bandwidth for later migrations.

In line 1 of Algorithm 1, the set representing the migration sequence is initialized. The current time for scheduling a migration is represented by t. In line 3, the set V_l contains the VMs that need to be migrated. This set can be computed based on the current and the targeted resource allocations, *i.e.*, $(y^{i0}_{n\bar{n}})_{i\in I, n\in N_i, \bar{n}\in\bar{N}}$ and $(y^{iT}_{n\bar{n}})_{i\in I, n\in N_i, \bar{n}\in\bar{N}}$.

The first step is to create the RIGs (line 6). First we create the set V_f containing the list of VMs that can be migrated at time t. Indeed, in some cases, it is not possible to directly migrate some of the VMs because the target machines at the current time may not have enough resources to host them. However, as both the initial and final resource configurations provided as inputs are supposed to be feasible, some VMs will be eventually moved from the target machine and thereby resources will be freed. In this case, the migrations that are not possible at the current time will be triggered subsequently. In order to create the RIGs, we first parse all the VMs in V_f. For each $v \in V_f$, we check if it is possible to include it in one of the existing RIGs (RIG j is denoted by G_j), otherwise a new RIG is created to contain v. A VM v is included in an existing group G_j if (1) its source and destination are not the same as any of the VMs belonging to G_j, and (2) it is possible to migrate v using a path that does not share any link with other paths used to migrate the VMs belonging to G_j.

In the second step, one of the RIGs is chosen to be migrated (denoted by G_{j^*}) at time t such that the following costs are minimized:

Algorithm 1 Heuristic algorithm for migration sequencing

Require: Initial mapping, $(y^{i0}_{n\bar{n}})_{i\in I, n\in N_i, \bar{n}\in\bar{N}}$
Require: Target mapping, $(y^{iT}_{n\bar{n}})_{i\in I, n\in N_i, \bar{n}\in\bar{N}}$
Output: Migration sequence, S_q

1: $S_q \leftarrow \emptyset$
2: $t \leftarrow 0$ {t represents the start time for migrations}
3: $V_l \leftarrow$ Set of VMs to be migrated
4: **while** $V_l \neq \emptyset$ **do**
5: $V_f \leftarrow$ Set of VMs that can be migrated at time t
6: {Creating RIGs}
7: $J \leftarrow 1$ {J: number of RIGs}
8: **for each** VM $v \in V_f$ **do**
9: $new\_group \leftarrow$ **true**
10: **for** $j = 1$ to J **do**
11: **if** v can be included in G_j **then**
12: $G_j \leftarrow G_j \cup \{v\}$
13: $new\_group \leftarrow$ **false**
14: **end if**
15: **end for**
16: **if** $new\_group =$ **true then**
17: $J \leftarrow J + 1$
18: $G_J \leftarrow \{v\}$
19: **end if**
20: **end for**
21: {Selecting a RIG to be migrated at time t}
22: $min\_cost \leftarrow \infty$
23: **for** $j = 1$ to J **do**
24: $cost \leftarrow \mathcal{C}_T(G_j)$ {Cost of migrating G_j (Eq. 22)}
25: **if** $cost < min\_cost$ **then**
26: $min\_cost \leftarrow cost$
27: $j^* \leftarrow j$ {G_{j^*} is selected}
28: **end if**
29: **end for**
30: $S_q \leftarrow S_q \cup \{< t, G_{j^*} >\}$
31: $V_l \leftarrow V_l \setminus G_{j^*}$
32: $t \leftarrow t + \mathcal{C}_W(G_{j^*})$
33: **end while**
34: **return** Migration Sequence S_q

– Migration time: The migration time ($\mathcal{C}_M$) for a group G_j is computed as follows:

$$\mathcal{C}_M(G_j) = \max_{v\in G_j} migration\_time(v)$$

where the migration time is calculated based on Equations (1) and (2).

– Waiting time: After selecting G_j to be migrated at time t, the scheduler has to wait for some time before starting the migration of the next group of VMs. We call this time the waiting time for the group G_j denoted by $\mathcal{C}_w(G_j)$. The idea is to start the migration of some VMs belonging to the subsequent RIG as soon as possible if they do not share any resources with the VMs being migrated.

– Impact on other RIGs' migration time: If we select to migrate G_j at time t, this operation will have an impact on the migration time of the other RIGs. The impact can be either positive (*i.e.*, decrease migration time) or negative (*i.e.*, increase migration time) based on the new residual bandwidth. This cost is computed as follows:

$$\mathcal{C}_I(G_j) = \sum_{k\in\{1\dots J\}\setminus j} \mathcal{C}_M(G_k/G_j) - \mathcal{C}_M(G_k) \qquad (21)$$

where $\mathcal{C}_M(G_k/G_j)$ is the migration time of the group G_k knowing that the migration of the group G_j has already been scheduled. This cost represents the expected gain or loss in migration time of the subsequent groups if G_j is scheduled first.

The total migration cost for RIG G_j is a weighted sum of the above mentioned costs. It can be written as follows:

$$\mathcal{C}_T(G_j) = \alpha\mathcal{C}_M(G_j) + \beta\mathcal{C}_W(G_j) + \gamma\mathcal{C}_I(G_j) \qquad (22)$$

where α, β, and γ are weights used to adjust the influence of each component. Note that all these costs have the same unit (seconds).

The RIG that minimizes the total cost, denoted by G_{j^*}, is selected to be migrated (line 27). The VMs of the group G_{j^*} are then added to the migration sequence and removed from V_l (line 31). The time t is increased by the amount of waiting time for G_{j^*} (line 32). The process is repeated until there are no VMs to be migrated (*i.e.*, $V_l = \emptyset$).

Finally, we analyze the running time of our algorithm. The grouping of VMs into RIGs (Line 8-20) requires $O(|V_l|)$ time to complete. The greedy migration sequence generation process (Line 22-32) takes at most $|V_l|$ rounds to complete. Each round runs in $O(|V_l|)$ time as it requires recomputing the migration cost of every remaining RIG and selecing the RIG with lowest cost for migration (Line 23-29). Thus, the total running time of the algorithm is $O(|V_l|^2)$.

VII. Evaluation

We evaluate the performance of CQNCR through extensive simulation. We generated four different VM deployment scenarios (Table I). For each scenario, VDCs, VMs and virtual links are generated. Their initial mapping is generated randomly whereas the target mapping is generated by consolidating VMs and virtual links as compactly as possible. In order to reach the target mapping, migration sequences are generated using two different algorithms: a baseline approach, and (ii) our proposed solution (referred to as CQNCR hereafter). We compare and contrast the obtained results to demonstrate the effectiveness of CQNCR.

In the following, we first describe the simulation setup and the deployment scenarios. We then present the metrics used to evaluate performance. Finally, we provide the comparison results between CQNCR and the baseline.

A. Simulation Setup

In our experiments, we simulate a data center with 1024 servers and 256 switches, connected according to a tree topology [5]. Each physical machine is configured with 16 CPU cores and 64 GB of memory and connected to a top-of-rack switch with 1 Gbps bidirectional link.

We perform simulations for the four different scenarios. For each scenario, table Table I shows the number of VDCs, VMs,

TABLE I
DEPLOYMENT SCENARIOS

Scenarios	# of VDCs	# of VMs	# of Vir. Links	Initial mapping		Target mapping		# of Migrations
				Active Phy. Hosts	Active Phy. Links	Active Phy. Hosts	Active Phy. Links	
$S-1$	3	13	14	13	57	3	10	13
$S-2$	10	95	72	95	307	14	30	95
$S-3$	50	488	183	485	1483	75	194	486
$S-4$	100	1045	360	935	2668	150	390	998

TABLE II
VM INSTANCES

EC2 Instance Type	vCPU	Memory (GB)
m3.xlarge	4	15
m3.2xlarge	8	30
m1.small	11	1.7
m1.medium	1	3.75
m1.large	2	7.5
m1.xlarge	4	15
c3.large	2	3.75
c3.xlarge	4	7
c3.2xlarge	8	15
c3.4xlarge	16	30
c1.medium	2	1.7
c1.xlarge	8	7
g2.2xlarge	8	15
cg1.4xlarge	16	22.5
m2.xlarge	2	17.1

virtual links, and migrations as well as the number of active physical machines (*i.e.*, hosting at least one VM) and active physical links (*i.e.*, used to embed at least one virtual link) for the initial and target mappings. The initial mapping is generated by randomly embedding VMs in the physical machines. The target mapping is generated by consolidating VMs of the same VDC as compactly as possible while minimizing the number of active servers and links. For instance, in scenario $S-1$, we consider 3 VDCs consisting of 13 VMs and 14 virtual links. In the initial mapping, 13 physical machines and 57 physical links are active. After the VM consolidation, only 3 physical hosts and 10 physical links are used. To reach the final mapping 13 migrations are required. In the other scenarios $S-2$, $S-3$, $S-4$, we consider a larger number of VDCs, VMs and virtual links, which leads to a higher number of migrations.

The requirements of VMs in terms of number of virtual CPUs (vCPUs) and memory size are selected from the Amazon EC2 instance specifications [2] as shown in Table II. The bandwidth of the virtual links are selected randomly between 250 and 750 Mbps. The image size of a VM depends on the deployed OS and applications. Hence, the image size of each VM is selected randomly between 1 GB to 100 GB. All our simulations are conducted on a machine with dual quad-core 2.4 GHz Intel Xeon E5620 processors and 12-GB of RAM.

Given a particular migration sequence, and the initial and final mappings, the simulator computes total migration time, average VDC downtime, and average VM downtime.

B. The Baseline Algorithm

We compare the performance of CQNCR with that of a baseline algorithm that performs parallel migrations at each time step while greedily decide the order of migrations. Specifically, this baseline algorithm starts by identifying the VMs that can be directly migrated. This means that (i) the destination machines have sufficient resources to host these VMs, and (ii) there is enough bandwidth to embed all virtual links connected to them. The algorithm uses disjoint paths to migrate in parallel the VMs with the shortest migration time. Next, it re-embeds the virtual links connected to the migrated VMs and updates the residual bandwidth of the physical links. This process is repeated until all VMs are migrated.

C. Results

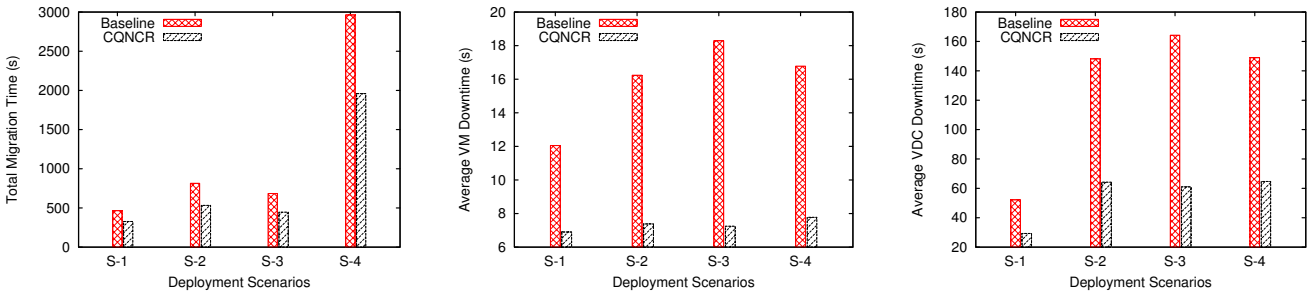
For each considered scenario, we evaluate three performance metrics:

- *Total migration time:* the total time to perform all VM migrations according to the migration plan.
- *VM downtime:* the duration of downtime incurred when the VM is migrated. This time corresponds to the duration of the stop-copy phase (Eq. 3).
- *VDC downtime:* the downtime of a VDC is the time span within which at least one of its VMs is down.

In our simulations, we set all the weighting factors in the total migration cost (Eq. 22) to 1 (*i.e.*, $\alpha = \beta = \gamma = 1$).

Fig. 5(a) reports the total migration time for the four considered scenarios. As we can see, in all scenarios CQNCR outperforms the baseline algorithm by a large margin. For scenario $S-1$, the baseline algorithm takes 465.5 seconds to migrate the VMs, whereas CQNCR takes only 327.73 seconds, which is around 30% improvement. In case of scenario $S-2$, all VMs are migrated within 814.6 seconds using the baseline and 533.28 seconds using CQNCR. Hence, CQNCR achieves around 35% improvement. Similarly, for scenarios $S-3$ and $S-4$ where the number of migrations is large (486 and 998, respectively), the improvement in total migration time is also around 36%.

The average downtime per VM is reported in Fig. 5(b). For the baseline algorithm, the average VM downtime for the four scenarios are 12.05, 16.24, 18.29, and 16.78 seconds, respectively. For CQNCR the downtimes are 6.9, 7.38, 7.25, and 7.78 seconds, respectively. Hence, in general, CQNCR reduces VM downtimes by up to 60%. Indeed, CQNCR ensures that the VMs are migrated at higher bandwidth and hence can significantly reduce the downtime.



(a) Total migration time (b) Average VM downtime (c) Average VDC downtime

Fig. 5. Total migration time and downtime considering all VDCs

Finally, Fig. 5(c) shows the average downtime per VDC. In scenario $S-1$, using the baseline algorithm, the VDC downtime is around 52 seconds, whereas it is around 29 seconds using CQNCR. It is clear from the figure that, for all scenarios, CQNCR reduces the downtime by up to 60%.

CQNCR performs significantly better than the baseline algorithm as it always groups together a collection of VM migrations (with the shortest combined migration time) that do not share any resources and performs these migrations in parallel. The second factor that allows CQNCR to further reduce the total migration time is the fact that it starts by selecting the VMs whose migration reduces the migration time of the subsequent groups as described in Section VI.

VIII. Conclusion

VM migrations are frequently used to optimize resource allocations and achieve several performance objectives in today's cloud environments. In this context, determining the optimal VM migration sequence is an important problem as it has a direct impact on both resource efficiency (in terms resources used for migration) and application performance (in terms of service downtime). In this paper, we presented an effective technique (called CQNCR) for determining the execution order of massive VM migrations within data centers. In particular, given an initial and a target resource allocation, CQNCR allows to find an efficient migration plan that minimizes the total migration time, the average individual VM downtime and the average VDC downtime. Simulation results show that, compared to the baseline, CQNCR improves total migration time by up to 35% and VM/VDC downtime by up to 60%.

References

[1] http://www.zdnet.com/amazon-web-services-suffers-outage-takes-down-vine-instagram-flipboard-with-it-7000019842/.

[2] http://aws.amazon.com/ec2/instance-types/instance-details/.

[3] S. Al-Haj and E. Al-Shaer. A Formal Approach for Virtual Machine Migration Planning. In *Proceedings of the* 9^{th} *International Conference on Network and Service Management (CNSM 2013)*, pages 51–58, 2013.

[4] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Towards predictable datacenter networks. In *Proceedings of ACM SIGCOMM*, August 2011.

[5] M. Bari, R. Boutaba, R. Esteves, L. Z. Granville, M. Podlesny, M. G. Rabbani, Q. Zhang, and M. F. Zhani. Data Center Network Virtualization: A Survey. *IEEE Communications Surveys Tutorials*, 15(2):909–928, 2013.

[6] R. Boutaba, Q. Zhang, and M. F. Zhani. Virtual Machine Migration: Benefits, Challenges and Approaches. In H. T. Mouftah and B. Kantarci, editors, *Communication Infrastructures for Cloud Computing: Design and Applications*, pages 383–408. IGI-Global, USA, 2013.

[7] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Networked Systems Design & Implementation*, NSDI, pages 273–286, 2005.

[8] R. Gandhi and J. Mestre. Combinatorial algorithms for data migration to minimize average completion time. In *Proceedings of the* 9^{th} *International Conference on Approximation Algorithms for Combinatorial Optimization Problems*, APPROX, pages 128–139, 2006.

[9] S. Ghorbani and M. Caesar. Walk the line: Consistent network updates with bandwidth guarantees. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 67–72, New York, NY, USA, 2012. ACM.

[10] 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 *Proceedings of the International Conference Co-NEXT*, 2010.

[11] F. Hermenier, J. Lawall, and G. Muller. BtrPlace: A Flexible Consolidation Manager for Highly Available Applications. *IEEE Transactions on Dependable and Secure Computing*, 10(5):273–286, 2013.

[12] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall. Entropy: a consolidation manager for clusters. In *Proceedings of the ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE, pages 41–50, 2009.

[13] M. R. Hines, U. Deshpande, and K. Gopalan. Post-copy live migration of virtual machines. *SIGOPS Oper. Syst. Rev.*, 43(3):14–26, July 2009.

[14] S. Lo, M. Ammar, and E. Zegura. Design and analysis of schedules for virtual network migration. *IFIP Networking*, 2013.

[15] V. Mann, A. Gupta, P. Dutta, A. Vishnoi, P. Bhattacharya, R. Poddar, and A. Iyer. Remedy: network-aware steady state VM management for data centers. In *Proceedings of the* 11^{th} *international IFIP TC 6 conference on Networking - Volume Part I*, pages 190–204, 2012.

[16] V. Mann, A. Vishnoi, A. Iyer, and P. Bhattacharya. VMPatrol: Dynamic and Automated QoS for Virtual Machine Migrations. In *Proceedings of the* 8^{th} *International Conference on Network and Service Management*, CNSM, pages 174–178, 2012.

[17] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[18] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif. Sandpiper: Black-box and gray-box resource management for virtual machines. *Computer Networks*, 53:2923–2938, December 2009.

[19] K. Ye, X. Jiang, D. Huang, J. Chen, and B. Wang. Live migration of multiple virtual machines with resource reservation in cloud computing environments. In *IEEE International Conference on Cloud Computing (CLOUD)*, pages 267–274, 2011.

[20] K. Ye, X. Jiang, R. Ma, and F. Yan. VC-Migration: Live Migration of Virtual Clusters in the Cloud. In *ACM/IEEE* 13^{th} *International Conference on Grid Computing (GRID)*, pages 209–218, 2012.

[21] M. F. Zhani, Q. Zhang, G. Simon, and R. Boutaba. VDC Planner: Dynamic Migration-Aware Virtual Data Center Embedding for Clouds. In *Proceedings of the* 13^{th} *IFIP/IEEE Integrated Network Management Symposium (IM 2013)*, Ghent, Belgium, May 2013.