Acinonyx: Dynamic Flow Scheduling for Virtual Machine Migration in SDN-enabled Clouds

Adel Nadjaran Toosi
Faculty of Information Technology
Monash University
Clayton, Australia
adel.n.toosi@monash.edu

Rajkumar Buyya
School of Computing and Information Systems
The University of Melbourne
Parkville, Australia
rbuyya@unimelb.edu.au

***Abstract*—Cloud providers rely on virtualization technology for efficient data center management and service delivery. Live Virtual Machine (VM) Migration is one of the critical features of this technology enabling cloud providers to relocate VMs between servers without disrupting applications currently running on it. One of the key challenges of VM live migration is that it creates elephant flows over the network links connecting the source to destination servers due to the transfer of the entire memory and possibly disks of the VM. In this paper, we leverage software-defined networking (SDN) and propose a dynamic flow scheduling approach called Acinonyx. Our solution is designed to minimize the negative impact of live VM migration on data center network traffic and reduce migration time in cloud data centers with multiple network paths between servers. Acinonyx adaptively installs flow entries on the switches of the shortest path with the lowest congestion and redirects VM migration traffic to the appropriate path. We describe the implementation of the proposed solution over our testbed environment, where we use OpenStack for managing commodity servers and OpenDaylight SDN controller for managing the OpenFlow switches. Our experimental results show that Acinonyx can significantly reduce the migration time of VMs while improving the overall network throughput for other VM communications.**

***Index Terms*—Software-defined Networking, Cloud Computing, *Dynamic Flow Scheduling*, *Live VM Migration*, OpenDaylight, OpenStack**

I. Introduction

Virtualization is the critical building block of the operation and maintenance for many modern data centers. Advances in virtualization contribute to increased adoption of Virtual Machines (VMs) for isolation, consolidation and migration purposes [1]. Amongst all significances of VMs, live migration is a promising and efficient means of relocating running VMs between servers (physical hosts) with no or minimum impact on the VM's availability. In cloud data centers, live VM migration is used as a management tool facilitating various functions such as hardware maintenance, load balancing, energy saving, and disaster recovery.

Despite all the benefits, live VM migration may have significant adverse impacts on data center network traffic. In fact, conventional live migration techniques typically transfer VM's CPU state, all memory pages, and disks from the source to the destination [2]. As a result, live migration of VMs may create elephant flows over the network links connecting the source to the destination causing network congestion for other applications sharing the same links. Moreover, network congestion may also impact the performance of live VM migration itself by increasing downtime and migration time.

To address these issues, some studies have focused on the network-aware planning of VM migration by determining the best time and sequence of VM migrations to minimize the overall migration time and the impact of migration overhead [3]–[5]. However, little or no attention has been paid to the selection of network paths for VM migration to avoid network congestion. The reason is that existing data center networks are optimized to statically select a single path across available paths for forwarding flow packets between each source/destination pair [6].

The emergence of software-defined networking (SDN) and its capabilities to shape and optimize network traffic from a logically centralized management controller provide opportunities for dynamic flow scheduling based on the network utilization and flow size in short time scales. Nowadays, the use of SDN for the resource management within and across datacenters has been suggested by many research proposals [7]–[10]. The tightly coupled control and data (forwarding) planes in traditional networking devices has been separated in SDN which allows us to efficiently program flow scheduling policies in a single point of management for an administrative domain such as a cloud data center [7].

In this context, we investigate if "it is possible to reduce live VM migration time and overhead by dynamically scheduling flows in a cloud data center having multiple paths between a given pair of physical hosts". To the best of our knowledge, this paper represents the first attempt towards this challenge.

Accordingly, in order to minimize the live VM migration time and overhead, we propose a migration orchestrator called *Acinonyx*[1] that dynamically pushes forwarding rules to the network switches using SDN controller APIs. Acinonyx collects active flows statistics from the SDN controller and installs appropriate flow entries for VM migration traffic in all constituent switches according to the opportunistic utilization of the residual (spare) bandwidth of the physical links.

[1]*Acinonyx* is a genus within the cat family with the only living species of the *Cheetah* (Acinonyx Jubatus) who is close to extinction. We selected this title to 1) raise awareness for this beautiful and endangered animal and 2) race towards the minimization of VM migration time like a cheetah.

We present a full implementation of Acinonyx on a real testbed prepared for this purpose [11]. Our proposed flow scheduling policy uses OpenDayLight APIs to dynamically install flow entries on physical switches in the network. We conduct experiments to show the effectiveness of our approach using OpenStack interfaces for live migration of VMs in our SDN-enabled testbed platform. The results show that Acinonyx is able to reduce the live VM migration time up to 12% and increase the total number of bits transmitted by other traffic in the network by 7%.

The rest of the paper is organized as follows: Section II provides the motivation for dynamic flow scheduling and the background for data center networks and VM migration. In Section III, we propose our dynamic flow scheduling algorithm. Section IV presents the system architecture and the implementation of the proposed flow scheduling algorithm. The testbed and experimental setup along with performance evaluation and experimental results are discussed in Section V. Section VI outlines related work. Finally, Section VII concludes the paper with directions for future work.

II. Background

In this section, we will describe (a) data center network architectures and common practices for routing traffic, (b) VM migration and its timing, and (c) a motivating example for dynamic flow scheduling of VM migration traffic.

A. Data Center Networks

As the scale of modern cloud data centers has been rapidly growing beyond tens of thousands of servers, establishment of networks providing sufficient bisection bandwidth in the data centers has become of paramount importance. In addition, the rise of VMs has also driven massive increases in "east-west" traffic between servers in the cloud data centers compared to large "north-south" traffic in enterprise data centers. The push for building such data center networks has promoted researchers and practitioner to explore horizontal expansion of networks leveraging many relatively inexpensive switches instead of using traditional tree topologies with high speed and high port density core switches [6]. These efforts have given birth to several alternative network topology designs such as VL2 [8], PortLand [12], and BCube [13]. These topologies create what is known as multi-rooted trees having a larger number of parallel paths between any given source and destination servers. Figure 1 illustrates a simple fat-tree topology [14] representing four equal-cost shortest paths between a given pair of servers. The figure shows two highlighted hosts and different paths that traffic between these two can traverse.

To take advantage of multiple paths, the current state of the art is to use Equal-Cost Multi-Path (ECMP[2]) routing [15]. ECMP is a forwarding mechanism that distributes traffic over multiple paths based on the hash of certain parameters of layer 3 headers. In this way, the load will be split across multiple paths, while packets of a given flow are guaranteed

[2]Analysis of an Equal-Cost Multi-Path Algorithm, https://tools.ietf.org/html/rfc2992

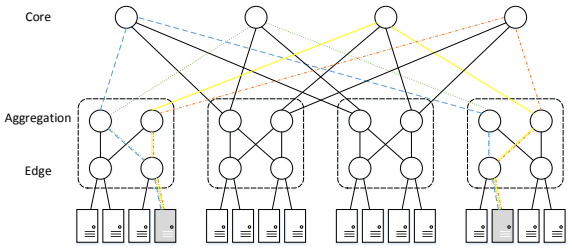


Fig. 1: Simple fat-tree network topology with four equal-cost shortest paths between a given pair of servers.

to take the same path and the arrival order at the destination is maintained. This is very important for the performance of applications, as out of order packet delivery significantly degrades the performance of TCP connections. Even though ECMP load-balancing is efficient in many cases, its static per-flow hashing can cause substantial bandwidth loss due to the collision of elephant flows such as live VM migration traffic. Thus, in this paper, we aim to exploit the high degree of parallelism and residual bandwidth available on the links to perform live VM migration in a shorter time scale through dynamic flow scheduling.

B. Virtual Machine Migration

Migrating VMs enables an administrator to move a VM instance from one host to another. A typical scenario is the case of planned maintenance on the source host or even performing disaster recovery. Moreover, VM migration can also be useful to balance system load when many VM instances are running on a specific overloaded host. On the contrary, VMs can be consolidated into a few servers in order to minimize resource wastage or energy consumption.

VM migration can be performed in two ways: Non-live (cold or simply migration) and live (hot) migration. In cold migration, the VM is shut down, then moved to another hypervisor and restarted on the destination. The application running on the VM is disrupted in this case. Live migration allows moving a running VM with no or minimal interruption to the application running on the VM as opposed to pure *stop-and-copy* of the cold migration. Even though our proposed method can be equally applied to cold migration, we focus on live VM migration here, because in public clouds often it is not possible or desirable to stop tenants' applications.

The "pre-copy" approach is the most common technique to live migrate a VM [5]. The hypervisor pre-copies the entire memory pages of the VM to the destination while the application is still running on the source. Then iterative pre-copy process is done in multiple rounds to transfer dirty pages (memory pages modified during the last pre-copy). Typically, there is a set of pages modified so often that pre-copy rounds will never finish without stopping the VM. Subsequently, at the final stage stop-and-copy is performed and the VM is resumed at the destination. If there is no shared-storage, live migration requires copying all disks from the source to the destination

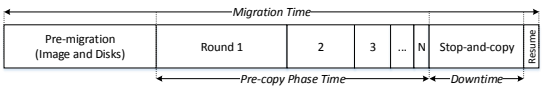


Fig. 2: Pre-copy live VM migration.

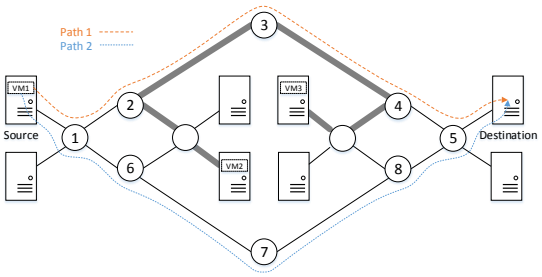


Fig. 3: Example of live VM migration when there are multiple paths and different bandwidth residual available.

host. Figure 2 schematically shows the pre-copy live migration process and its timing.

Mann et al. [9] presented a mathematical model of live VM migration in KVM and VMware. Based on their model, duration of pre-copy and stop-and-copy phases is computed as:

$$T = \frac{M \times \frac{1-(R/L)^n}{1-(R/L)}}{L}, \quad (1)$$

where M is the memory size of the VM, R is the VM dirty page rate and L is the amount of bandwidth available. n stands for the number of pre-copy phases calculated as the minimum values of two other equations omitted for the sake of brevity here. Note that Equation (1) shows that the pre-copy and stop-and-copy phases time non-linearly depends on the dirty page rate and available bandwidth.

C. Motivation

In this section, we provide a motivating example to show the impact of dynamic flow scheduling on the VM migration time. Figure 3 depicts a network topology of the small-scale data center that consists of 10 switches and 8 hosts. We assume that links in the topology have a capacity of 100Mbps and the thickness of each link shows traffic (bandwidth usage) on the link. Moreover, we assume that there exists an active flow between VM2 and VM3 using 50% of the bandwidth (50Mbps) on the connecting links.

In our example, the system administrator wants to migrate VM1 from the *source* host to the *destination*. As shown in Figure 3, there are two shortest paths available between the source and the destination, namely *path 1* (Switches 1,2,3,4, and 5) and *path 2* (Switches 1,6,7,8, and 5). We assume that the static flow routing or the ECMP hashing function, by default, has chosen *path 1* for the VM1 live migration. Now, suppose that the total size of data that must be transferred during the migration is fixed and equal to 5GB. Since *path 1* and

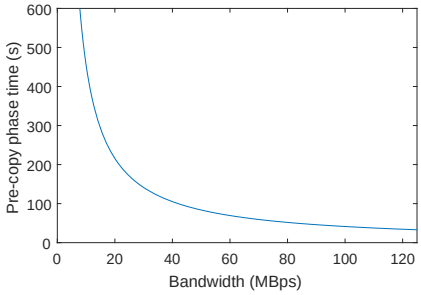


Fig. 4: Pre-copy phase time versus bandwidth based on Equation (1), when M = 4GB, R = 1MBps, and n = 4.

the active flow between VM2 and VM3 share two links, the effective bandwidth which can be used for the VM migration on *path 1* is reduced to 50Mbps. If the flow between VM2 and VM3 remains active on the same bit rate and the status of the network is not updated during the entire migration, the total migration time of VM1 will be 5GB/50Mbps=819.2 seconds. However, if *path 2* was initially selected for the migration, the migration time would be reduced by 50% to 409.6 seconds. This motivates us to employ dynamic flow scheduling to opportunistically exploit bandwidth residuals on multiple paths between servers in multi-rooted trees network topologies.

Contrary to what is shown above, the performance gain in this example is not linearly proportional to the available bandwidth. In our example, we assumed that the total size of data to be transferred is fixed no matter which path is chosen. However, as discussed in Section II-B, the duration of pre-copy phase is non-linearly proportional to the available bandwidth if dirty page rate of a VM is not zero. For instance, Figure 4 plots pre-copy phase time versus available bandwidth if the memory size of the VM is 4GB, dirty page rate is 1MBps and the number of pre-copy rounds equals to 4 according to Equation (1). As shown in the figure, when the available bandwidth increases, the time of pre-copy phase exponentially decreases. Thus, increasing bandwidth reduces the migration time not only explicitly through a higher transmission rate but also implicitly by fewer amounts of data that must be transferred.

III. Acinonyx Dynamic Flow Scheduling

Our proposed dynamic flow scheduling is shown in Algorithm 1. The key insight of our approach is to iteratively redirect the live VM migration traffic on a path with the lowest load when multiple shortest paths are available between the source and destination. This reduces the VM migration time and better utilizes available residual bandwidth on the links. In an SDN-enabled data center, this can be performed by a flow setup on the switches along the path. The SDN controller configures flow entries and propagates them to switches.

The input to the algorithm is a pair of IP addresses representing the source (s) and the destination (d) hosts for the migration. The algorithm is executed per VM migra-

tion and as long as the VM migration is in progress. The GETTOPOLOGY() method at Line 2 exploits the SDN controller APIs to obtain the network topology. Then, it finds all available shortest paths between the source and the destination. Since our topology is an unweighted graph, a simple breadth-first search (BFS) can be used for finding the shortest paths in this case. To avoid the time complexity of BFS, one can store all possible paths between all given pairs once and use it when required. This is especially suitable for data centers in which the network topology updates are infrequent.

The outer loop at Lines 5-18 iterates over all shortest paths between the source and the destination to find a path that has the highest residual bandwidth on its most used link (highest byte rate or most congested). That is, the inner loop at Lines 7-13 finds the link with highest byte rate along the path (most used), while the outer loop iterates over paths and selects the path with minimum byte rate on its most used link. Since live VM migration traffic certainly happens through one of the shortest paths, we have to make sure that the migration traffic is excluded from the calculation. Thus, GET-BYTE-RATE at Line 8 determines the byte rate for the matching VM migration flow, f, which is subsequently deducted from the link byte rate, b, at Line 9. As soon as the the best path between the source and the destination is found, appropriate flow rules are pushed into the switches to redirect migration traffic to this path (Line 21). In order to make sure that the migration traffic does not bounce between paths with small differences, we check that the difference between the byte rate of the most used link on the current path and the newly suggested one is more than a certain *switching ratio* (α) of the current one as shown in Line 20. MAXBYTERATE($currentPath$) finds the byte rate at the most used link of the current path similar to the calculation in the inner loop at Lines 7-13. At Line 23, the algorithm waits for a *sleeping interval* (β) before the next flow scheduling round.

The time complexity of the shortest path algorithm is $O(|V| + |E|)$ where $|V|$ is the number of nodes (vertices) and $|E|$ is the number of links (edges) in the topology. The outer loop in the algorithm iterates over all possible shortest paths and the inner loop goes over all the links on the path. Therefore, the time complexity of this part can be expressed as $O(|P| \times |p|)$, where $|P|$ is the number of equal-cost shortest paths and $|p|$ is the path length. In practice, three-tier data center network architecture with multi-rooted trees, the maximum of $|p|$ is 6 links and $|P|$ depends on the number of core switches. The overall time complexity of *Acinonyx* is $O(|V| + |E|)$ as the $|P| \times |p|$ equals to $|E|$ in the worst case. It is worth mentioning that Acinonyx is highly scalable in large scale data centers as a separate run of the algorithm can executed for each live VM migration flow.

One may wonder why the proposed dynamic scheduling algorithm cannot be generalized to all other flows in the data center. The reasons are twofold: 1) network traffic workloads characterized by many small, short-lived flows would gain limited benefit from dynamic scheduling [6]. VM migration traffic is one of the limited elephant flows that cloud provider has the full knowledge of its long duration. 2) The proposed algorithm is designed to opportunistically exploit residual bandwidth on multiple available paths. A dynamic flow scheduling that manages all flows together requires more complex algorithms with a global management that hinders the scalability of such scheduler.

Algorithm 1 Acinonyx Dynamic Flow Scheduling

Input: s, d

```
1:  while MIGRATION-IS-IN-PROGRESS() do
2:      G ← GET-TOPOLOGY()
3:      P ← FIND-SHORTEST-PATHS(G, s, d)
4:      min ← +∞
5:      for p in P do
6:          max ← 0
7:          for link in p do
8:              (b, f) ← GET-BYTE-RATE(link)
9:              r ← b − f
10:             if r > max then
11:                 max ← r
12:             end if
13:         end for
14:         if max ≤ min then
15:             path ← p
16:             min ← max
17:         end if
18:     end for
19:     mbr ← MAXBYTERATE(currentPath)
20:     if mbr − min > mbr × α then
21:         PUSH-FLOWS(path)
22:     end if
23:     SLEEP(β)
24: end while
```

IV. SYSTEM ARCHITECTURE AND DESIGN

To enhance the use of Acinonyx in real cloud environments, we propose and design a system prototype based on today's cloud architecture and technologies. Figure 5 illustrates the high-level architecture of the proposed framework integrating our dynamic flow scheduling technique.

Our implementation relies on the VM management of cloud operating systems (platforms) such as OpenStack. Although here we limit our discussion to OpenStack, Acinonyx can be incorporated to any other similar platforms, since no modified features of OpenStack have been used in this implementation. OpenStack uses *libvirt*, a popular toolkit for managing virtual machines and virtualization functionalities, as a default driver. With a VM to be migrated, OpenStack calls libvirt APIs to perform the migration. Following that, libvirt takes care of live VM migration and provides updates to OpenStack. Once transferring memory and disk content of the VM is done, the virtual network infrastructure gets updated and all traffic is redirected to the newly started VM. All networking elements updated in this process are limited to software switches of the virtual network infrastructure managed by OpenStack. It

is worth mentioning that OpenStack is in charge of the entire VM migration process (see Section II-B) and Acinonyx only perfoms the network flow scheduling for the VM migration traffic including the entire migration data transferred between the source and the destination hosts.

To use our proposed dynamic scheduling, it suffices that Acinonyx is notified regarding the initiation of the VM migration and the source and destination hosts. This can be done manually by the system Admin or OpenStack notifies Acinonyx. From this point onwards, Acinonyx autonomously monitors the network status and updates flow entries using the northbound APIs of the SDN controller (e.g., OpenDayLight) and iteratively reroutes VM migration traffic between libvirt agents to the lowest cost path. Every time a route change is essential, Acinonyx sends requests to the SDN controller to push required flow entries matching the VM migration traffic to network switches. To perform this, SDN controller uses OpenFlow [16] protocol for pushing match-action flow entries to manage the forwarding planes on the switches. Note that Acinonyx does not interact with Neutron (the OpenStack networking as a service component), since the flow scheduling is only performed on the physical switches (Core, Aggregation, Edge switches in Figure 5). There is no need to configure sofware switches managed by OpenStack as each software switch is essentially connected to a single physical edge switch. Acinonyx monitor the status of the VM migration process (running, completed) by checking it via OpenStack APIs. Once the VM migration is completed, the Acinonyx dynamic flow scheduling process is stopped. Each live VM migration in the system is managed independently and requires execution of a separate process of Acinonyx.

V. Performance Evaluation

To evaluate the performance of our proposed method, we conduct experiments in our real-world testbed for SDN-enabled cloud computing [11] and report measurements. We have created our own experimental testbed since production systems such as Amazon Web Services (AWS) or even private clouds such Australian Nectar Cloud[3] will not allow users to access and modify their low-level infrastructure elements such SDN controllers and physical switches needed for our experiments. In the following, we briefly discuss our experimental testbed, while more details can be found in [11]. Then, we discuss our experimental setup including traffic generation and default routing mechanism used for the experiments. Finally, we present results of experiments. The goal of experiments is to determine how our proposed method affects the VM migration time. We also invistigate the impact of parameters such as *switching ratio* (α) and *sleeping interval* (β) on the performance of the algorithm with various background traffic.

[3]https://nectar.org.au/research-cloud/

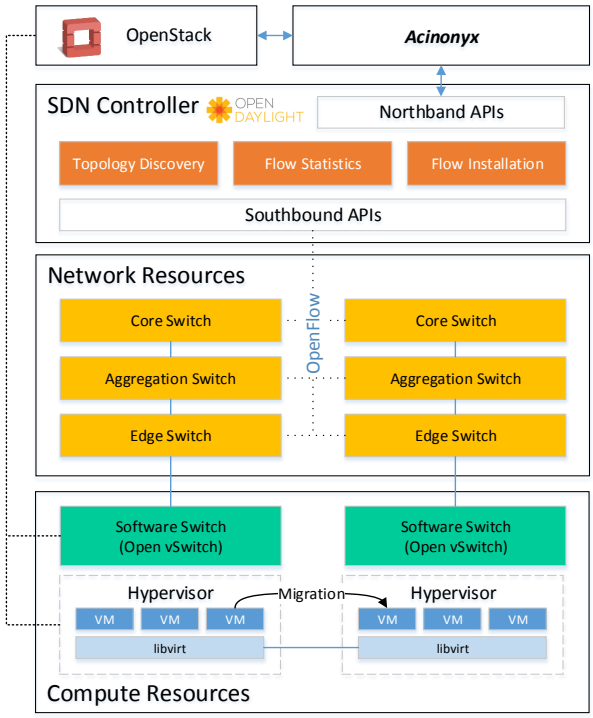


Fig. 5: The high-level system architecture.

A. Experimental Testbed

To build our testbed, we use our existing machines to set up a cluster of 8 heterogeneous servers with the specifications shown in Table I. Our experimental network is built based on the fat-tree reference topology which creates a multi-rooted network with multiple paths between servers. However, for a minimal 3-layer fat-tree topology [14], we require at least sixteen server machines. Thus, we resort to a resembling physical network topology shown in Figure 6. In order to minimize the cost of network equipment, we use *Raspberry Pis* (Pi 3 Model B), low-cost embedded computers, to host Linux-based *Open vSwitch (OVS)* [17]. Each Raspberry Pi with integrated OVS plays the role of a 4-port switch (4 USB ports with USB-to-Ethernet adapters) with an external port for control (the built-in Ethernet interface). The highest nominal bandwidth can be achieved on each USB 2.0 port is 480Mbps. However, here, the bandwidth is limited to 100Mbs since we used TP-Link UE200 USB 2.0 to 100Mbps Ethernet adapters.

Openstack (*Ocata* release) is used to manage our cloud platform. It allows for running and live migration of VMs in the testbed. The live VM migration in OpenStack can be performed through shared storage-based live migration or block migration. Block migration requires copying disks from the source to the destination host and takes more time and puts more load on the network. We use block migration in our experiments since VM disks are not shared between source and destination hosts. The OpenStack setup in our

TABLE I: Specifications of machines in the testbed.

Machine	CPU	Cores	Memory	Storage
2 x IBM X3500 M4	Intel(R) Xeon(R) E5-2620 @ 2.00GHz	12	64GB (4 x 16GB DDR3 1333MHz)	2.9TB
4 x IBM X3200 M3	Intel(R) Xeon(R) X3460 @ 2.80GHz	4	16GB (4 x 4GB DDR3 1333MHz)	199GB
2 x Dell OptiPlex 990	Intel(R) Core(TM) i7-2600 @ 3.40GHz	4	8GB (2 x 4GB DDR3 1333MHz)	399GB

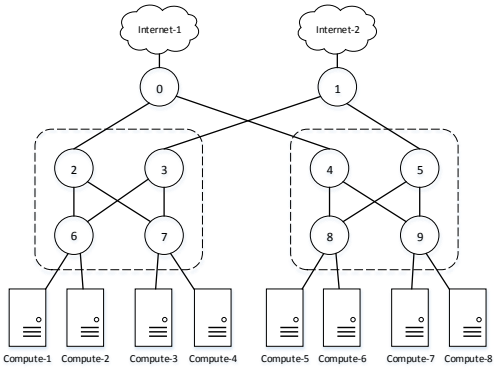


Fig. 6: The experimental network topology.

testbed uses *KVM/QEMU*[4] hypervisors and *libvirt* APIs to run and manage VMs on the host machines. Since libvirt uses TCP port range of 49152-49215 as destination ports for live migrations, we install layer 3 flow entries based on this range and source/destination IP addresses to perform flow scheduling. In practice, tagging/labeling, VLAN, or VXLAN tunneling methods can also be used to differentiate live VM migration traffic from other traffic in the network.

We used *OpenDaylight* (ODL)[5], a popular open-source SDN controller, as the SDN controller to handle our OpenFlow capable switches connected through an Out-of-Band (OoB) network. The SDN-controlled data network carries the live VM migration flows and the traffic between communicating VMs. ODL uses OpenFlow protocol to communicate with the Pi switches to set forwarding rules through its southbound API. Likewise, Acinonyx uses ODL northbound APIs to install flow entries on the switches or query global state information such as network topology, the number of packets and bytes processed by switch ports or flows.

B. Traffic Generation and Routing

In order to evaluate our method, we need to generate random and synthetic background traffic in our testbed. To achieve this, we used *Iperf3*[6] tool. A script is written to continuously start and stop *Iperf* clients and servers on randomly selected pair of hosts and send synthetic TCP or UDP flows between them. In our experiments, we refer to each traffic flow generated in this way as a *connection* and label it as *Conn-x* (eg. *Conn-1*).

In the data network, there exists a default static route for any given communicating pair of hosts. In the default static routing flow entry setup, it is assured that for any given pair of hosts that are connected to the same switch, different links are used on the edge switch for sending traffic to upper layer switches in the topology. Acinonyx pushes higher priority flow entries to the switches for live VM migration traffic to override the default path.

[4]Kernel-based Virtual Machine (KVM), https://www.linux-kvm.org/.

[5]ODL, https://www.opendaylight.org/.

[6]Iperf, https://iperf.fr.

TABLE II: Background traffic with *Iperf* tool in TCP mode for the 1st experiment. BW stands for bandwidth in Mbps.

Conn-x	Time	Source	Destination	Length	BW	Path
Conn-1	0	Compute-5	Compute-4	60s	10	8-5-1-3-7
Conn-2	25	Internet-1	Compute-2	120s	30	0-2-6
Conn-3	40	Compute-4	Compute-5	120s	60	7-3-1-5-8
Conn-4	125	Compute-8	Compute-2	120s	40	9-4-0-2-6
Conn-5	180	Internet-1	Compute-6	30s	50	0-4-8
Conn-6	185	Compute-5	Compute-8	60s	20	8-5-9
Conn-7	200	Compute-2	Compute-8	90s	40	6-2-0-4-9

TABLE III: The migration time in seconds and the average number of bits transmitted in megabits between *Iperf* agents when Acinonyx or Static Routing is used.

Metric	Static Routing	Acinonyx
Migration Time (s)	287	**256**
Average Throughput (Mbs)	32.0	**34.4**

C. Experimental Results

We carried out four experiments evaluating (a) the improvement in the total VM migration time and throughput, (b) the impact of flow scheduling frequency (sleeping interval), (c) the impact of switching ratio on the performance of the proposed method, and (d) the performance of multiple migrations.

1) Migration Time and Throughput Improvement: In the first experiment, we demonstrate that our proposed method improves the total VM migration time compared to the static routing mechanism. We generate a background traffic using *Iperf* in TCP mode as shown in Table II. The "time", "length", "bandwidth", and "path" columns in the table represent the time that connection starts, the duration that traffic remains active, target bandwidth in Mbps, and the list of switches that traffic flows through, respectively (see Figure 6 for the labels of switches). We conduct a live VM migration in OpenStack to move an idle `m1.small`[7] VM with `Ubuntu-16.04` image from *Compute-1* to *Compute-7* (see Figure 6) which starts at time time zero. The migration traffic can traverse through either of the two available shortest paths with switches 6-2-0-4-9 or 6-3-1-5-9. The byte rate for the links are measured in a 2-second interval and α and β are set to 0.4 and 5s, respectively. Each experiment is carried out three times and the mean value is reported.

Table III shows the migration time in seconds and the average throughput among *Ipref* agents when we use Acinonyx

[7]OpenStack `m1.small` flavor: 1 VCPUs, 2GB RAM, 20GB Disks

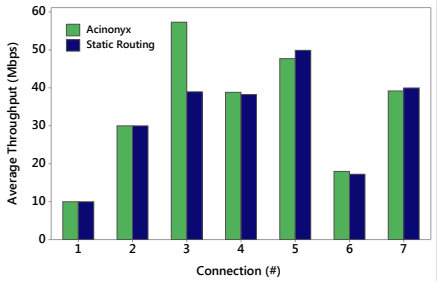
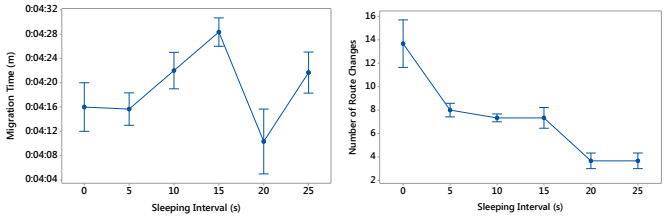


Fig. 7: Average throughput for each background connection in Table II when Acinonyx or Static Routing is used.



(a) (b)

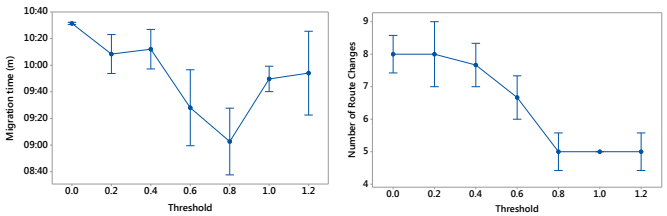
Fig. 8: The impact of Acinonyx sleeping interval (β) on (a) the migration time and (b) the number of route changes. Error bar represents one standard error from the mean.

and static routing. As results indicate, Acinonyx can reduce the migration time by 12% for the sample traffic shown in Table II. This shows that the Acinonyx dynamic flow scheduling can improve VM migration time by opportunistically rerouting the VM migration flow on the path with the lowest congested link.

Acinonyx not only reduces the VM migration time but also provides better load balancing and consequently allows for a higher network throughput for other VM communications. Figure 7 shows the throughput between every *Iperf* client/server connection in Mbps when Acinonyx and static routing are used. Even though in cases of Conn-5 and Conn-7, a marginally lower throughput could be achieved using Acinonyx, the overall average throughput significantly increases from 32.0 Mbps to 34.4 Mbps representing 7% improvement as shown in Table III. In particular, the average throughput for connection 3 is substantially increased when Acinonyx is used (roughly 47% increase). The reason is that it shares the same path with the VM migration traffic from *Compute-1* to *Compute-7* in the static routing configuration.

2) Impact of the Sleeping Interval (β): In the second experiment, we evaluate the impact of flow scheduling frequency on the performance of the live VM migration. We repeat experiment 1 and migrate the VM from *Compute-1* to *Compute-7* while we vary the sleeping interval between 0 and 25 with 5-second steps. The background traffic and all other settings are the same as in the previous experiment.

Figure 8 depicts the migration time and the number of flow rescheduling when the sleeping interval varies. As the figure shows the best average migration time of 4 minutes



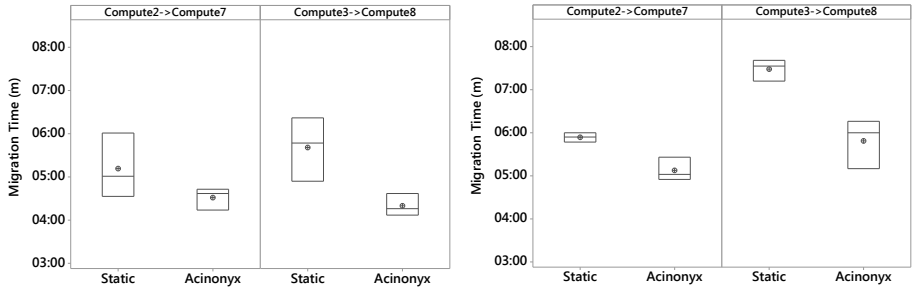
(a) (b)

Fig. 9: The impact of Acinonyx switching ratio (α) on (a) the migration time and (b) the number of route changes. Error bar represents one standard error from the mean.

and 10 seconds can be attained when the sleeping interval is set to 20 seconds. The total number of route changes, in this case, is equal to 3.6 on average. Results show that the best value for sleeping interval strongly depends on the background traffic. More frequent route and quick route switching has an adverse impact on the performance of TCP flows due to the arrival order of packets going through different paths [18]. The main point is that the optimal sleeping interval for Acinonyx depends on the type and the nature of background traffic. In fact, a longer sleeping interval delays the required flow scheduling, and a smaller sleeping interval makes our method sensitive to short-lived flows. In practice, the impact of sleeping interval is marginal as you can see in the experiment. As long as the sleeping interval is short enough which allows path switching during the migration process, the application of Acinonyx is beneficial.

3) Impact of Switching Ratio (α): In the third experiment, we evaluate the impact of switching ratio (α) on the performance of the live VM migration. We migrate an `m1.small` VM from Compute-8 to Compute-1 and generate background traffic but this time in UDP mode. The reason we used a different background traffic is to show that our method can perform well under various background traffic. A total of 14 connections between random pairs of source and destination are generated with uniform random numbers in the range of $(0, 250)$ for the start time of the connection, bitrates in range $(10, 50)Mb$, and duration in the range of $(0, 250)$ seconds. α is varied from 0 to 1.2 to evaluate its impact on the migration time and the number of route changes. The byte rate for the links are measured in a 5-second interval, and the sleeping interval β is set to 10s.

The results of the experiment are shown in Figure 9. In this experiment, the VM migration took 11 minutes and 8 seconds on average using the default static routing. Figure 9a shows that the minimum migration time is achieved when the ratio is set to 0.8. Correspondingly, the mean number of route changes for different values of the ratio is shown in the Figure 9b. As expected, the number of route changes decreases when a larger ratio value is set. However, as can be seen in the figure, after a certain point, increasing ratio leads to a higher migration time. Our experimental results show that better results can be achieved if the ratio is set appropriately. In practice, the



(a) No Background Traffic. (b) With Background Traffic

Fig. 10: Box plots of migration time for two simultaneous migrations when *Static* and *Acinonyx* flow scheduling are used. $\oplus$ shows mean value.

optimal setting of switching ratio depends on the traffic and characteristics of the network. The network administrator can set the value based on the nature of traffic in the data center and some preliminary experimentation.

4) Multiple Migrations: Acinonyx is designed to work in a totally distributed fashion. A separate process of the algorithm is executed to perform dynamic scheduling for each VM migration. To evaluate the performance of the algorithm in case of multiple VM migrations that happen regularly in large cloud data centers, we conduct two experiments here in which a pair of VM migrations coincides. We perform live migration of a given pair of `m1.small` VMs, one from *Compute-2* to *Compute-7* and the other from *Compute-3* to *Compute-8*.

In the first experiment, we do not generate background traffic. As shown in Figure 10a, Acinonyx on average reduces the migration time by 13% and 24% for the migrations from *Compute-2* to *Compute-7* and *Compute-3* to *Compute-8*, respectively. Since in this experiment, migration flows share multiple links along the path based on the default static routing, Acinonyx immediately detects the condition and schedules flows on different routes to maximize throughput. There is a small chance that Acinonyx processes monitoring shortest paths to find a route with the highest spare capacity fall into the trap of redirecting their flow within a short time to the same path back and forth. This issue can recurrently happen if the scheduling interval of processes perfectly matches. A simple solution to avoid this condition is to set different sleeping intervals for dynamic flow scheduling processes.

In the second experiment, we repeat the same experiment when there is background traffic with a mixture of UDP and TCP connections. A total of 11 randomly generated UDP and TCP connections between different pairs of source and destination is used in this scenario. Figure 10b depicts box plots of migration time for every VM migrations. Results show that, similar to the previous experiment, Acinonyx reduces the migration time for both VM migrations. In comparison to no background traffic, there is a slight increase in migration time for all cases due to the existing network traffic.

VI. Related Work

Virtualization technology has been the cornerstone of resource management and optimization in cloud data centers for the last decade. Many research proposals have been expressed on the basis of live VM migration to conduct maintenance, load balancing, energy saving, and disaster recovery. Accordingly, there is a large body of literature focused on improving the efficiency of live migration mechanism [19]. However, little attention has been given to the impact of live VM migration on the data center network or how network resources can efficiently be utilized for live VM migration.

Bari et al. [5] propose a method for finding an efficient migration plan. They try to find a sequence of migrations to move a group of VMs to their final destinations while migration time is minimized. In their method, they monitor residual bandwidth available on the links between source and destination after performing each step in the sequence. However, their method does not consider scheduling new routes for the VM migration traffic. Similarly, Ghorbani et al. [3] propose an algorithm to generate an ordered sequence of VMs to migrate and a set of forwarding state changes. While we focus on the dynamic scheduling of migration flows on the lowest cost path, they concentrate on imposing bandwidth guarantees on the links in a way that no link capacity can be violated at any point during the migration. The VM migration planning problem is also tackled by Li et al. [4]. They address the workload-aware migration problem in which they propose methods for selection of candidate virtual machines, destination hosts, and sequence for migration. Similar to [3], [5], they do not consider updating network switches for efficient utilization of available network paths. All these studies focus on the order of migration for a gorup of VMs while taking into account network cost. However, in our work, we adapt the network and forwarding states of switches to efficiently use available resources and residual bandwidth of the links.

SDN decouples the network control and data forwarding planes and enables the optimization and shaping of the network traffic in a programmable and centrally manageable way. This provides great opportunities for traffic engineering mechanisms specialized to the requirements of resource management in data centers, for example, dynamic flow scheduling for the VM management. Al-Fares et al. [6] propose a generic dynamic flow scheduling system that tries to compute non-conflicting paths for flows and instructing switches to reroute

traffic accordingly. The complexity of their method is significantly high as they try to schedule all running flows in the data center. This reduces the applicability of their method in real world data centers. However, we only focus on the dynamic flow scheduling specially designed for the live VM migration traffic that can be quickly applied in practice. There are many other efforts in the literature trying to exploit SDN features for efficient and reliable VM management. Cziva et al. [7] present an orchestration framework to exploit temporal network information to live migrate VMs and minimize the network-wide communications. Wang et al. [20] propose a VM placement mechanism to reduce the number of hops between communicating VMs, save energy, and balance the network load. Remedy [9] relies on SDN to monitor the state of network and estimate the cost of VM migration. Their technique detects congested links in the network and migrates a set of VMs to remove congestion on those links. Contrary to their method, we prevent congestion by rerouting migration traffic to the links with lower traffic.

VII. Conclusions and Future Work

Live VM migration is frequently used in cloud data centers for the efficient management of resources. However, Live migration itself creates large network traffic in cloud data centers affecting the network performance. In this paper, we proposed a dynamic flow scheduling technique in the pursuit of efficient use of available bandwidth for live VM migration in SDN-enabled cloud data centers with multiple paths between the given pair of servers. We showed that our technique results in more efficient live VM migration reducing the migration time up to 10% compared to existing static routing solutions. We also demonstrated the feasibility of our proposed technique by building a working prototype over a practical testbed using OpenStack as a cloud operating system and OpenDaylight as an SDN controller.

In the future, we plan to explore the impact Acynonix on the flows of applications running in the migrating VM. We also plan to extend our approach for efficient flow scheduling of combined multiple migrations. We will also explore deployment of applications with different workloads (e.g., stream processing, data analytics, web applications, scientific workflows) exhibiting various network traffic characteristics and study the performance of the proposed approach.

Acknowledgments

This work was partially supported by Australian Research Council (ARC) Discovery Project (Grant no: DP160102414).

References

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, ser. SOSP '03. New York, NY, USA: ACM, 2003, pp. 164–177.

[2] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, ser. NSDI'05. Berkeley, CA, USA: USENIX Association, 2005, pp. 273–286.

[3] 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*, ser. HotSDN '12. New York, NY, USA: ACM, 2012, pp. 67–72.

[4] X. Li, Q. He, J. Chen, K. Ye, and T. Yin, "Informed live migration strategies of virtual machines for cluster load balancing," in *Proceedings of the 8th IFIP International Conference on Network and Parallel Computing*, ser. NPC'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 111–122.

[5] M. F. Bari, M. F. Zhani, Q. Zhang, R. Ahmed, and R. Boutaba, "Cqncr: Optimal vm migration planning in cloud data centers," in *Proceedings of 2014 IFIP Networking Conference*, June 2014, pp. 1–9.

[6] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 19–19.

[7] R. Cziva, S. Jout, D. Stapleton, F. P. Tso, and D. P. Pezaros, "Sdn-based virtual machine management for cloud data centers," *IEEE Transactions on Network and Service Management*, vol. 13, no. 2, pp. 212–225, June 2016.

[8] 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 *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, ser. SIGCOMM '09. New York, NY, USA: ACM, 2009, pp. 51–62.

[9] 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*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 190–204.

[10] A. Sadasivarao, S. Syed, P. Pan, C. Liou, I. Monga, C. Guok, and A. Lake, "Bursting data between data centers: Case for transport sdn," in *2013 IEEE 21st Annual Symposium on High-Performance Interconnects*, Aug 2013, pp. 87–90.

[11] A. N. Toosi, J. Son, and R. Buyya, "CLOUDS-Pi: A low-cost raspberry-pi based micro data center for software-defined cloud computing," *IEEE Cloud Computing*, 2018, (Accepted on 12th of July).

[12] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "Portland: A scalable fault-tolerant layer 2 data center network fabric," in *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, ser. SIGCOMM '09. New York, NY, USA: ACM, 2009, pp. 39–50.

[13] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "Bcube: A high performance, server-centric network architecture for modular data centers," in *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, ser. SIGCOMM '09. New York, NY, USA: ACM, 2009, pp. 63–74.

[14] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, ser. SIGCOMM '08. New York, NY, USA: ACM, 2008, pp. 63–74.

[15] C. Hopps, "Analysis of an equal-cost multi-path algorithm," United States, 2000. [Online]. Available: https://tools.ietf.org/html/rfc2992

[16] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[17] N. Networks, "Open vSwitch: An open virtual switch," http://openvswitch.org/, Sept. 2017.

[18] R. Carpa, M. Dias De Assuncao, O. Glück, L. Lefèvre, and J.-C. Mignot, "Evaluating the Impact of SDN-Induced Frequent Route Changes on TCP Flows," in *Proceedings of the 13th International Conference on Network and Service Management*, ser. CNSM'17, Tokyo, Japan, Nov. 2017.

[19] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, *Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 254–265.

[20] S.-H. Wang, P. P. W. Huang, C. H. P. Wen, and L. C. Wang, "Eqvmp: Energy-efficient and qos-aware virtual machine placement for software defined datacenter networks," in *The International Conference on Information Networking*, ser. ICOIN2014, Feb 2014, pp. 220–225.