Early Notification and Dynamic Routing: An Improved SDN-Based Optimization Mechanism for VM Migration

Xuanlong Qin[1], Dagang Li[1](✉), Ching-Hsuan Chen[2], and Nen-Fu Huang[2]

[1] Peking University Shenzhen Graduate School, Shenzhen, China
xuanlong@pku.edu.cn, dgli@pkusz.edu.cn
[2] National Tsing Hua University, Hsinchu, Taiwan
napolenon0sam@gmail.com

Abstract. Virtual Machine (VM) live migration is a crucial practice commonly applied in data centers, which can help users and enterprises manage network services more flexibly and economically. In general, large enterprises always require uninterrupted network service in VM migration, but the VM's own behavior and limited link resources often result in increased transmission times. We found that VMs may have two types of problems in live migration: bursty traffic and packet loss in the tail of the migration round, both of which tend to cause performance loss in the process of iterative migration. Thus, we propose an early notification (E-N) algorithm that can deal with these two problems. Simultaneously, considering more flexible software-defined networks, we put forward a dynamic routing (D-R) algorithm to ensure the link performance of VM live migration. Finally, we present experiments prove that VM migration performance can be improved by combining E-N algorithm and D-R algorithm.

Keywords: Live migration · Software-defined network
Transport layer · Migration performance

1 Introduction

With the development of network techniques, virtualization technology has been widely used on modern computing platforms. VM technology allows a customer with multiple parallel tasks to replace multiple servers with a single server. In addition, VM migration can reallocate and schedule resources. VM migration technology has attracted considerable interest for data center management and cluster computing in recent years. Representative products include XenMotion and Vmotion, which were implemented as built-in tools for virtualization platforms. In data centers, VM migration is often combined with software-defined

This work was supported by Science, Technology and Innovation Commission of Shenzhen Municipality (JCYJ20160525154348175, ZDSYS201603311739428).

networks (SDNs), because SDN characteristics can separate network into control plane and data plane, making traffic more transparent and easier to distinguish. This improves the load balance of system, increases the fault tolerance of the system, and provides various other benefits.

1.1 Background

Live VM migration is one of the key enabling technologies for fault tolerance and load balancing, they use the migration time in exchange for a lower downtime. For some services, a downtime of one second may result in approximately 1100 dollars of lost revenue [5]. So How to minimize the total migration time and downtime is a crucial research topic.

At present, numerous well-known practices focus on hosts for VM live migration; such practices include compressing memory [4], removing memory redundancies, establishing a Writing Work Set, establishing a path with a routing algorithm, dynamic resource assignment, and managing the order of VM transfers. However, few people have precisely optimized the resource consumption of the transport layer and link layer. In this paper we put forward a method combining SDNs and transport layer optimization to improve the performance of VM migration.

Currently, VM migrations are usually executed at data centers, using connections with high bandwidth and low latency. A typical network is fat tree structure with shared links. Any congestion that occurs during data transmission causes extremely serious link congestion and incurs an exorbitant cost. Because SDNs are flexible enough to guarantee excellent performance, the trend of VM migration involves SDN technology in data centers has become ubiquitous.

1.2 Our Contribution

This paper presents a Transmission Control Protocol (TCP) approach to optimize VM migration from the perspective of data transmission. The proposed approach solves the problem of traffic bursts and packet loss during the tails of the migration rounds (Packet lost at the memory iteration copy phase). This paper puts forward a method of SDN-based link optimization and proposes a routing strategy to reduce link congestion and total migration time.

The remainder of this paper is structured as follows. Section 2 presents precopy live migration and SDN dynamic resource assignment. In Sect. 3, we analyze the problems that arise in live migration. In Sect. 4, we propose an algorithm to solve these problems and list its pseudocode. In Sect. 5, we test the correctness of our VM and SDN models; then evaluate the performance of our proposed algorithm in Sect. 6. Finally, we conclude our work in Sect. 7.

2 Related Work

To clarify the optimization mechanism for VM live migration, we introduce precopy migration and SDN dynamic resource assignment in this section.

2.1 Precopy Live Migration

The precopy algorithm proposed by Clark [1], have proven to be remarkably effective tools for enabling data center management in a non-disruptive manner. Xen, KVM, and VMware all use precopy algorithms for live migration; they all use a memory-to-memory approach as depicted in Fig. 1.

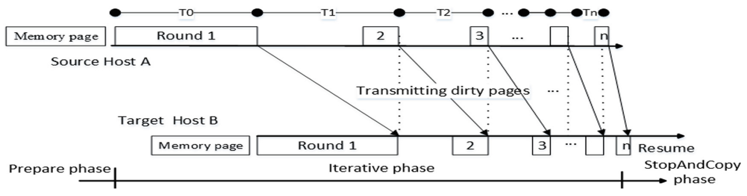

Fig. 1. Precopy timeline

Different hypervisors implement precopy migration differently, but most hypervisors use the same basic pattern of three phases: the preparation phase, the iterative phase, and the stop-and-copy phase. During the preparation phase, the main responsibility of the application is the preparation of relevant resources before the migration starts. During the first round of the iteration phase, the host transfers all the memory pages, at this time the VM is still working, thus some memory pages must be overwritten (the overwritten pages are usually called dirty pages). The dirty pages must be retransferred to the destination in the next round. After several rounds of synchronization, VM arrives stop-and-copy condition and enters the stop-and-copy phase. When the VM resumes service on the destination host, indicates the migration is complete. Unlike static replication algorithms, a precopy method optimizes downtime to avoid unforeseen overhead and errors caused by long VM downtimes.

The preparation phase generally requires a fixed amount of time to execute procedures and to allocate resources. The stop-and-copy phase typically requires approximately 200 ms of downtime. We focus on optimizing the iteration stage.

Xen hypervisor requires four conditions before it enters the stop-and-copy phase: (1) The speed at which memory pages are dirtied exceed the memory transfer speed. (2) The dirty page to be transferred next round reach a certain threshold. (3) The number of rounds executed within the iteration phase exceed a given value. (4) The total migrated data must be larger than three times the VM memory size.

When the stop-and-copy conditions have been satisfied, the VM is shut down on the source host and the remaining dirty pages are transferred to the destination host.

2.2 SDN Dynamic Resource Assignment

SDNs technology is mainly used in traffic engineering and it can also benefit VM migration. When multiple VMs must be migrated at the same time, with a centralized view of the network, it is possible to compute the bandwidth required for each migration [6]. By minimizing the correlations between the migration schemes of different VMs, we can maximize the migration bandwidth and thus we can minimize the total migration time and the downtime [7]. Furthermore, topology-aware live migration can reduce the communication costs of pairwise VMs [8] and increase the overall throughput.

An SDNs controller can obtain various types of network information and control that information to optimize the flexibility of the data center network in response to traffic challenges.

3 Problem Statement

3.1 Bursty Traffic in the Migration Rounds

In the previous section, we described the precopy algorithm of VM live migration. Precopy migration captures dirty pages through a shadow page table and marks these dirty pages with a bitmap. In each round of the iteration phase, the bitmap determines what data must be transmitted in the current round, what data must be transmitted during next round, and what data must be transmitted in the last round. After all data of one round has been sent, the bitmap is flushed. After bitmap designates data to be transmitted this round, Xen transmits the designated dirty pages batch by batch; these dirty pages are copied to the TCP buffer in the kernel through socket functions, then transmitted by TCP services. At the same time, the socket functions record the amount of data; when data transmission is completed this round, Xen evaluates whether the stop-and-copy condition has been reached. If the iteration phase continues, then the bitmap starts a new round, scans the data, and transfers the next set of dirty pages. This process produces a short round interval; it is almost negligible, but Xen cannot see the behavior of TCP after the data has been copied to the TCP buffer.

In general, the interval between one round and the previous round is so short that the TCP receiver does not send a window update message to the sender when a round finished. Therefore, at any given round, the TCP sending window usually maintains the status of the previous round. Because the previous migration data has already been transferred, no data packets are in the sending window for transmission. Thus, when the next round's data is ready to be transferred, the TCP sender usually sends all the packets of data that the sending window can hold to the network at once. Network congestion is probable.

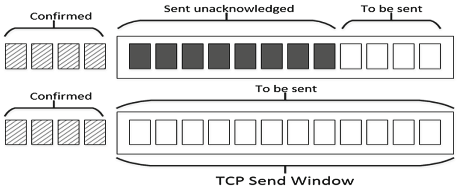

Fig. 2. Change of cwnd in live migration

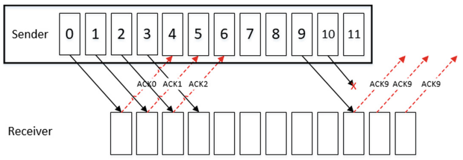

Fig. 3. Packet loss in the tail

Figure 2 illustrates the state of the standard TCP sending window. The data packets of the TCP sending window are divided into two categories: those that have been sent but remain unacknowledged and those that remain to be sent. To achieve flow control, the sender can use the notification window passed by receiver to adjust the size of the sending window. During the iteration phase, when the previous round has been transferred, the TCP sending window is empty. When the next round of transmission starts, all packets that can be contained within the sending window are usually sent to the network at the same time.

Considering the business value and cost of performance, most commercial switches are shallow buffer switches. The sudden flows of traffic required for VM migration can easily cause a switch's cache to overflow, resulting in a loss of packets and network congestion. With the number of lost packets increased, the TCP congestion algorithm enters its congestion avoidance phase even slow-start phase, thus increases both the total migration time and the amount of data transferred. Once congestion occurs, especially in the stop-and-copy phase, system downtime increases, and the cost is unacceptable.

3.2 Packet Loss in the Tail of the Migration Rounds

TCP congestion control is divided into four segments: slow start, congestion avoidance, fast retransmission, and fast recovery. When TCP in congestion avoidance phase and the TCP sender is continuously receiving three duplicate ACKs for one packet, TCP executes its fast retransmission algorithm and retransmit this packet immediately without waiting for the retransmission counter to expire.

In live VM migration, the data packets are transmitted round by round; if the last packet for a round is transmitted successfully and confirmed by receiver's ACK, then the data of that round has been completely transmitted. But if the system is transmitting the last three packets, and if one of the data packets is lost, this round has no data packet can be used to send ACK. The result is TCP can only wait for the retransmission counter to run out.

This case as shown in Fig. 3, the TCP receiver thinks that the network congestion is serious and need re-entering the slow-start phase. The initial window value of the slow start was default set to be 1, which means that at least two rounds of transmission are required for the final three packets.

However, the actual situation may be that the network traffic is not sufficiently heavy to cause the TCP system to enter its slow-start phase. The TCP sender cannot send enough duplicate ACKs because the application layer cannot provide enough data to send. Generally, this problem of packet loss at the end of the transmission also happens in standard TCP connections, but is amplified in live migration because the precopy task requires many rounds and this risk occurs at the end of each round. Re-entering the slow-start phase lengthens the transmission time, and increases the amount of data transferred.

3.3 Link Contention Problem

In data centers, the network topology usually contains redundant links to provide fault tolerance. In the present work, redundant links provide extra bandwidth in addition to their failover function [3]. We chose not to implement a spanning tree protocol because it could have blocked several ports.

Consider a fat tree topology as shown in Fig. 4. Assume each link in the topology has the same capacity. Three VMs migrate simultaneously; three source hosts are paired with three destination hosts. Without load balancing, all migration pairs would choose one of the upper switches as a route. For example, they might pass through switches (5, 1, 4) at the same time, resulting in link contention, as shown in Fig. 4. In this scenario, some of the links have noteworthy collisions whereas other links are idle. With load balancing, such contention can be avoided, as shown in Fig. 5. Ideally, the transfer speed would be three times higher since the migration pairs are not required to share a single link, and two redundant links are fully utilized.

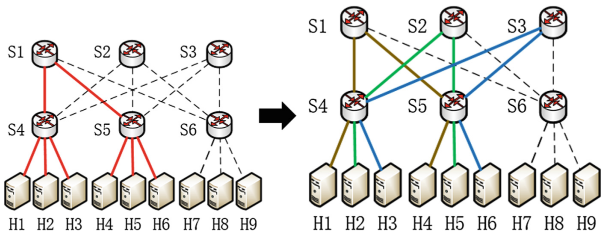

Fig. 4. Multiflow single path

Fig. 5. Multiflow multipath

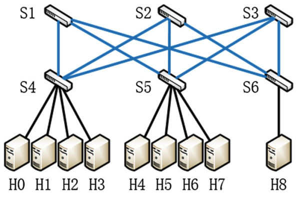

Fig. 6. Experiment topology

The difference between the existing load balancing mechanisms of legacy networks and our SDN dynamic routing is that the traditional mechanisms do not consider the characteristics of VM migration, and VM migrations cannot retrieve usage information from the underlying network. Also, VM migrations do not occur evenly among hosts so different link loads typically occur.

4 Algorithm

4.1 E-N Algorithm

For the precopy task, we propose an early notification (E-N) algorithm to optimize the performance of VM live migration.

The algorithm counts how many packets have been sent. When this round of data has been completely sent, the E-N algorithm notifies the application layer to prepare the next round's data immediately, instead of waiting for all the packets that have been sent to be acknowledged at this round. This algorithm is equivalent to eliminate the interconnection between the rounds.

Because the time required to scan the dirty page bitmap is short, when the next round of data is ready to be sent, the data in the TCP sending window has not yet been sent. At that moment, TCP has sufficient data waiting to be sent that TCP can avoid the problem caused by packet loss in the tail of the migration round. Thus TCP can avoid entering its slow-start mode. Our E-N algorithm also solves bursty traffic problems in the migration rounds. When two rounds of data have been generated without any pause, TCP can use the standard sliding window protocol to expand or shrink its windows, and thus can avoid the aforementioned bursty traffic problem. The pseudocode for the E-N algorithm is as follows:

Algorithm 1. Early Notification Algorithm

```
1: **Input:** $V_{mem}$, **Output:** $T_i$, $V_i$
2: Connect(); sending a SYN packet
3: Last_turn_seq := iss; T_last_turn := now();
4: if idle() then
5:    if seqno + datalen < curseq then
6:       T_current := now()
7:       Ti := T_current - T_last_turn;
8:       Vi := seqno + datalen - last_turn_seq;
9:       VM_model(Ti, Vi);
10:      Senddata()
11:   end if
12: end if
```

V_{mem} represents the size of a memory page, T_i is the time of transmission for one iteration, V_i is the amount of data in that iteration, *iss* is the current sequence number, *curseq* is the sequence number of the last byte provided by the application.

Our E-N algorithm was executed when the TCP connection is established; VM record initial sequence number. If the sum of the sequence number and the length of the data is greater than curseq, which means that this round's data has been sent.

In addition, the proposed approach is more feasible and compatible than standard TCP. For ordinary TCP flows, the application layer does nothing and the transport behavior is as same as standard TCP. No hardware support is required; a simple interaction between TCP and host improve the performance of VMs live migration.

4.2 Dynamic Routing Algorithm

We implemented a dynamic routing mechanism using Ryu as the Openflow controller. Ryu is a component-based SDN framework that is concise and agile.

The controller constructs a graph using the detected network topology. Every time new communication pair enters the system, no matching happens in the flow table; a packet-in is required. Then the controller obtains the source IP and the destination IP and maps them to the nodes in the graph. With a source, a destination, and a graph, the controller is capable of computing the shortest path using Dijkstra's algorithm. After choosing a best route, its weight will be set to a maximum value to avoid multiple connection selecting the same optimal route in an update period.

The controller queries the statistics of each port for all switches every five seconds. We define the weight of each edge as the sum of the statistics over five seconds for the two ports connected by the edge. Because every new communication pair is routed along the least busy links, the total VM migration time can be minimized.

5 Test Model

We built a VM live migration model, implemented our algorithms in that model, validated the correctness of the model, and verified the performance of the algorithms.

5.1 VM-Based Model

We selected a VM performance and energy prediction model as proposed in [2] as our experimental model, and output the value of the congestion window to verify the behavior of the VM for the problem we proposed.

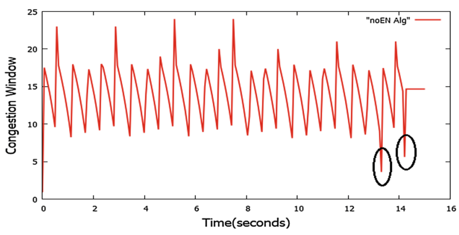

Fig. 7. Size of the congestion window

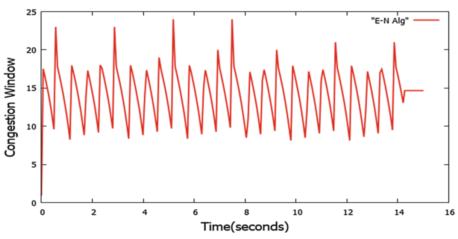

Fig. 8. Congestion Window after E-N algorithm

Figure 7 displays a horizontal axis to represent the time of migration, and displays a vertical axis to represent the size of the congestion window. For a system without our algorithm, the congestion window suddenly become smaller as indicated by dotted black ellipses. We examined the time at which the window fluctuated and found that the windows shrank during a round with bursty traffic and packet loss in the tail. The result of a system with our algorithm is illustrated in Fig. 8; the bursty traffic and packet loss were eliminated.

5.2 SDN Test Model

We used Mininet, a famous network emulation tool, to test the link contention problem. The topology is shown in Fig. 6. Three transmissions occurred in parallel: 4 to 1, 5 to 2, and 6 to 3. Without dynamic routing, the transmissions experienced severe link contention. As Fig. 9 shows, all transmissions were influenced. We implemented our dynamic routing mechanism, and Fig. 10 shows that it was notably effective. All transmissions were able to proceed at full speed.

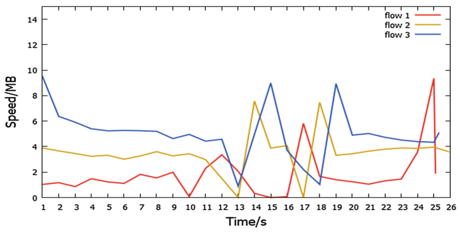

Fig. 9. Without dynamic routing

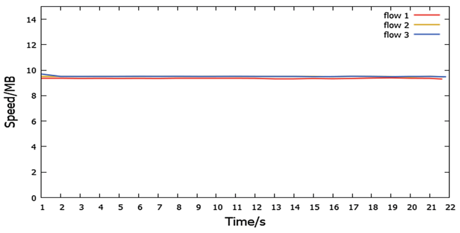

Fig. 10. With dynamic routing

6 Experiment Environment

In this section, we evaluate our E-N algorithm and D-R algorithm through a comparison experiment; the main performance indicator is total migration time.

The experiment is presented in two parts. Firstly, we introduce the basic parameters of the experiment. Then we analyze the experimental performance.

6.1 Experiment Setup

We conducted the experiment on one physical machine, with a four-core Intel i7 3.50 GHz processor, 16 GB RAM, and a 500 GB SATA hard disk. The physical machine ran Ubuntu 14.04 with the Linux 4.4.0-79 kernel; the hypervisor was virtualbox. The guest OSes were Ubuntu 14.04 with the Linux 4.4.0-31 kernel.

We use Mininet to construct network topology. The experiment topology was a simple fat tree that shown in Fig. 6. The bandwidth was set to 500 Mbps, link delay was set to 25 s, and the size of the VM was set to 128 MB, 256 MB, 512 MB, and 1024 MB for different trials. The dirty pages rate was set to 5000 and 10000 dirty pages (4 KB/page) to represent high and low rates of dirty pages.

NS3 is a famous network simulator used in some scientific papers; we implemented the VM's base model in NS3 as an individual host. With our setup, NS3 was able to send packets to a real network through the real network interface of its host and was identified by the Ryu controller. We set the VM stop-and-copy conditions by using the default parameters of Xen. The iteration round was 30. The stop-and-copy threshold was set as 200 KB. The transmitted data was not permitted to exceed three times the VM memory size.

6.2 Performance of Algorithm

Our experiment was divided into two parts, without and with background flow. The first part of this experiment verified the effectiveness of our algorithms for VM migration at different dirty page rates without background flow.

Because no background flow occurred, the probability of packet loss was reduced. In this condition, our total migration time was still reduced 0.5–2 s as Fig. 11 indicates.

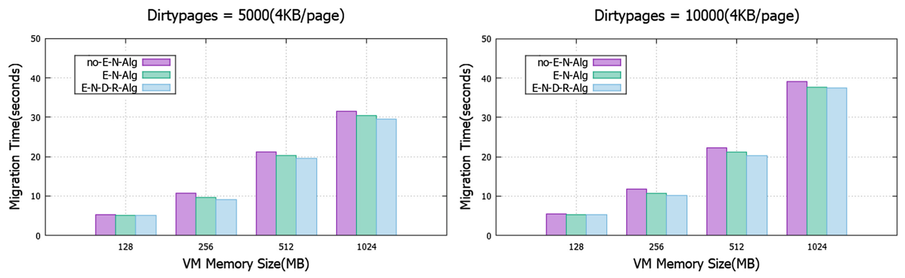

Fig. 11. No background flow

The second part used the same configuration with background flow and the result is illustrated in Fig. 12.

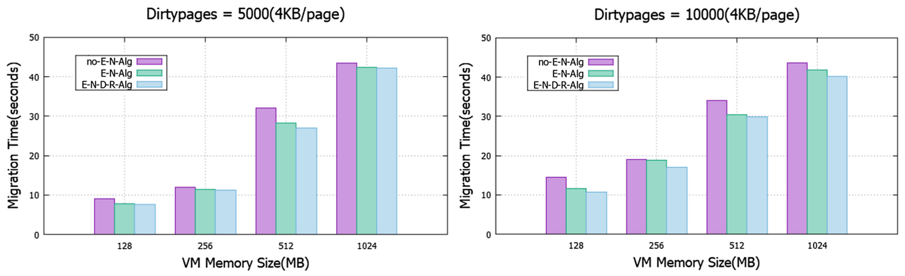

Fig. 12. With background flow

A high dirty page rate and the presence of background flow tend to enlarge the problem we explained in Sects. 3.1 and 3.2. A high dirty page rate and the presence of background flow tend to increase the benefits of using our algorithms.

With background flow, the total migration time is reduced about 2–3 s or more. If hundreds of migrations were done simultaneously, the optimization results would be more significant. The experimental results indicate that our algorithms are effective.

7 Conclusion

This paper uses the D-R and E-N algorithms to optimize the performance of VM live migration. It solves two problems: bursty traffic and packet loss at the tail. These problems often occur during VM migration. Our algorithms can minimize the total time for VM migration and improve performance.

References

1. Clark, C., Fraser, K., Hand, S.: Live migration of virtual machines. In: Proceedings of the 2nd conference on Symposium on Networked Systems Design and Implementation, NSDI 2005, Berkeley, USA, vol. 2, pp. 273–286 (2005)
2. Liu, H., Xu, C. Z., Jin, H., Gong, J., Liao, X.: Performance and energy modeling for live migration of virtual machines. In: Proceedings of Cluster Computing, vol. 16, pp. 249–264 (2013)
3. Liu, J., Su, L., Jin, Y., Li, Y.: Optimal VM migration planning for data centers. In: Proceedings of IEEE Global Communications Conference, pp. 2332–2337 (2014)
4. Jin, H., Wu, S., Shi, X.H.: Live virtual machine migration with adaptive, memory compression. In: IEEE International Conference on Cluster Computing and Workshops, Louisiana, USA (2009)
5. Salfner, F., Troger, P.: Downtime analysis of virtual machine live migration. In: 4th International Conference on Dependability, France, August 2011
6. Wang, H., Li, Y., Zhang, Y.: Virtual machine migration planning in software-defined networks. IEEE Trans. Cloud Comput. **PP**(99) (2017)
7. Yao, X., Wang, H., Gao, C.: VM migration planning in software-defined data center networks. In: IEEE 18th International Conference on High Performance Computing and Communication, Sydney, Australia (2017)
8. Cziva, R.: SDN-based virtual machine management for cloud data centers. IEEE Trans. Netw. Serv. Manag. **13**, 212–225 (2016)