VMPatrol: Dynamic and Automated QoS for Virtual Machine Migrations

Vijay Mann * , Anilkumar Vishnoi * , Aakash Iyer * Parantapa Bhattacharya †,
* IBM Research - India
Email:{vijamann,avishnoi,aakiyer1}@in.ibm.com
†Indian Institute of Technology, Kharagpur, India
Email:parantapa@cse.iitkgp.ernet.in

***Abstract*—As more and more data centers embrace end host virtualization and virtual machine (VM) mobility becomes commonplace, we explore its implications on data center networks. Live VM migrations are considered expensive operations because of the additional network traffic they generate, which can impact the network performance of other applications in the network, and because of the downtime that applications running on a migrating VM may experience. Most virtualization vendors currently recommend a separate network for VM mobility. However, setting up an alternate network just for VM migrations can be extremely costly and thus presents a barrier to seamless VM mobility. Therefore, it is apparent that VM migrations should be orchestrated in a network-aware manner with appropriate QoS controls such that they do not degrade network performance of other flows in the network while still being allocated the bandwidth they require for successful completion within the specified time lines.**

In this context, we present VMPatrol - a QoS framework for VM migrations. VMPatrol uses a cost of migration model to allocate a minimal bandwidth for a migration flow such that it completes within the specified time limit while causing minimal interference to other flows in the network. Our implementation and experimental evaluation of VMPatrol on real and virtual software testbeds demonstrates that automated bandwidth reservation can reduce the impact of migrations on other flows in the network to a negligible level.

***Keywords*-network aware VM migration; VM migration traffic modeling; automated QoS for VM migrations; virtualization**

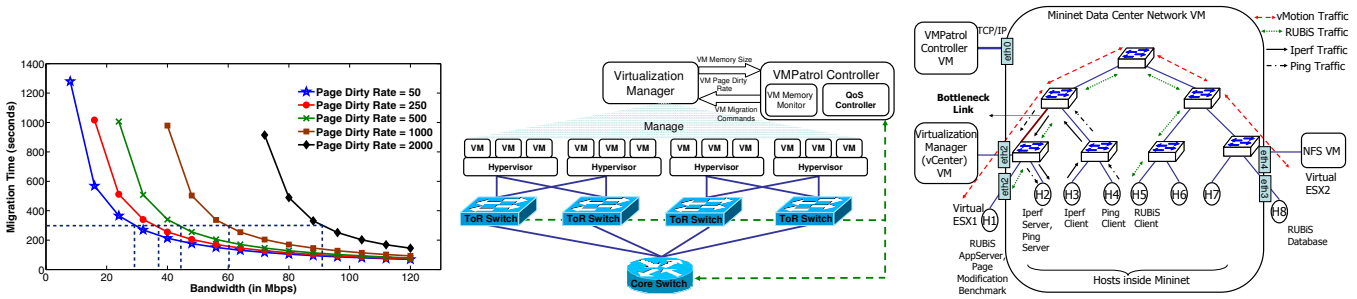
I. Introduction

As more and more data centers embrace end host virtualization and virtual machine (VM) mobility becomes commonplace, it is important to explore its implications on data center networks. VM migration is known to be an expensive operation because of the additional network traffic generated during a migration, which can impact the network performance of other applications in the network, and because of the downtime that applications running on a migrating VM may experience. Most virtualization vendors currently recommend a separate network for VM mobility [1], [2]. However, setting up an alternate network just for VM migrations can be extremely costly and thus presents a barrier to seamless VM mobility. Therefore, it is apparent that VM migrations should be orchestrated in a network-aware manner with appropriate QoS controls such that they do not degrade network performance of other flows in the network while still being allocated the bandwidth they require for successful completion within the specified time lines. We further argue that with appropriate QoS controls, network-aware VM migrations can be a useful tool to decongest network hot spots.

978-3-901882-48-7

Recent versions of hypervisors and virtual switches [3], [4] provide a static rate control mechanism for migration traffic at the end hosts based on user specified rates. These solutions suffer from two main drawbacks. First, it is difficult for a user to specify a rate for migration traffic without knowing the behavior of all the applications that might run on given VM. An easier option for a user is to specify the time by which a particular migration should always finish, regardless of the nature of the applications running on a VM. Second, rate control mechanisms on end hosts can only throttle a VM migration flow. In the absence of an end-to-end QoS policy that is enforced on all network elements on the migration path, it is still possible that a VM migration flow may not get the the bandwidth it requires to finish within a given time because of competing flows in the network. However, host based rate limiting solutions along with frameworks that allow a single QoS policy to be effectively applied on all switches across the network [5], [6] together provide the tools required to provide dynamic and automated QoS for VM migrations. An automated and dynamic QoS framework should take into consideration application behavior and ensure that migrations do not degrade network performance of competing flows, while allocating them sufficient bandwidth to finish within the specified time line.

In this context, we present the design and implementation of VMPatrol - a QoS framework for VM migrations. To the best of our knowledge, VMPatrol is the first system to provide dynamic and automated QoS for VM migrations. VMPatrol exploits a cost of migration model to predict link bandwidths required to migrate a given VM such that migration finishes within the specified time line. VMPatrol configures a minimal bandwidth on a network path (a set of virtual and real switches) for a given VM migration flow based on a cost of migration model and then orchestrates a VM migration on that path. VMPatrol ensures that the migration completes within the specified time limit while causing minimal interference to other flows in the network. Our experimental evaluation of VMPatrol demonstrates that automated bandwidth reservation for VM migration can reduce the impact of migrations on other flows in the network to a negligible level, while ensuring that the VM migrations finish within the specified time limit. In the absence of automated bandwidth reservation for VM migration, these flows faced up to 20% degradation in average



(a) Predicted migration times for a 1 GB VM and selection of a bandwidth value based on a migration deadline (330 seconds)

(b) VMPatrol architecture

(c) Virtual testbed used for evaluating VMPatrol

Fig. 1.

bandwidth and latency and a 60-70% increase in variance during a migration.

The rest of this paper is organized as follows. Section II describes the design and architecture of VMPatrol. We provide an experimental evaluation of VMPatrol in Section III. An overview of related research is presented in Section IV. Finally, we conclude the paper in Section V.

II. Methodology and Architecture

In this section we present the methodology and architecture of VMPatrol.

A. Methodology

VMPatrol builds on a cost of migration model that we proposed in [7]. VMware vMotion and KVM live migration both use pre-copy live migration technique. The following Theorem (taken from [7]) gives the equations for the number of pre-copy cycles, and for the total traffic generated during a migration, under the assumption that the following parameters are given constants: the memory size M of a VM in MB, the page dirty rate R of a VM in MB/s, and the bandwidth of the link used for migration L in MB/s. T is the user specified switchover goal time (equal to length of stop copy phase) and X is the user specified minimum required progress amount (in terms of reduction in memory being transferred in successive pre-copy iterations in MBs).

Theorem 2.1: The number of pre-copy cycles $n = \min\left(\left\lceil \log_{R/L} \frac{T \cdot L}{M} \right\rceil, \left\lceil \log_{R/L} \frac{X \cdot R}{M \cdot (L-R)} \right\rceil\right)$, and the total traffic generated by the migration $N = M \cdot \frac{1-(R/L)^{n+1}}{1-(R/L)}$.

We also note that the total migration time is:

N/L. **(1)**

and the time $W(L)$ spent on the stop-copy transfer (which is equal to the downtime experienced by an application) for a given migration bandwidth L, is given by:

$W(L) = \frac{M}{L} \cdot \left(\frac{R}{L}\right)^n$. **(2)**

We have validated the above model for VMWare hypervisor on a virtual testbed [7] and for KVM on a real hardware testbed (graphs omitted due to lack of space).

Figure 1(a) shows the predicted migration times for a VM with size 1 GB for different page dirty rates using equation 1 above. These page dirty rates correspond to typical workloads such as OLTP database benchmarks (3000 pages/second) and Quake 3 server (500 pages/second) as reported in [8]. Assuming a migration deadline of 330 seconds, the minimum bandwidth required is shown for different page dirty rates.

B. Architecture

VMPatrol architecture is shown in Figure 1(b). VMPatrol has been implemented in Python as a NOX [9] module as well in Java as a FloodLight module, both of which are popular OpenFlow controllers. VMPatrol has two main components: VM Memory Monitor, and QoS Controller.

VM Memory Monitor: VM Memory Monitor fetches the memory size and the current average page dirty rate for each VM. The settings for memory size are usually exposed by the virtualization manager. We use VMware ESX hypervisor and KVM as our virtualization platforms. VMware ESX hypervisor has a remote command line interface (RCLI) [10], that has commands to fetch memory size and also to initiate a migration. However, it does not expose a direct metric to measure page dirty rate. We currently rely on VMware ESX logs to find out page dirty rates. KVM provides a dirty page log facility, which provides user space with a bitmap of modified pages since the last call.

QoS Controller: QoS controller creates two queues on each port on all the switches in the network at startup. The first queue is a "default" queue and the second queue is a dedicated queue for migration traffic (the "migration" queue). By default, all traffic goes through the default queue. When a migration flow is detected, VMPatrol calculates the minimum bandwidth required for migrating a given VM (with a given memory size and page dirty rate) within a specified time line on a given path with a given total available bandwidth, as per the cost of migration model in section II-A. The path for migration is determined in a way such that the total available bandwidth of the path is higher than the minimum bandwidth required for the migration to finish within the deadline. VMPatrol then sets the bandwidth (min and max rates) of the "migration" queue to the calculated minimal value. It also installs an OpenFlow rule to "Enqueue" (using action OFPAT_ENQUEUE in OpenFlow 1.0) all migration traffic (identified by the specific port numbers that are used for migration between the source and destination end hosts) to the "migration" queue. This rule can also be installed at startup on all switches by specifying only the port numbers.

Creation of the queues is not part of the current OpenFlow protocol. A separate OF config protocol has been proposed for creation and configuration of these queues. Currently, VMPatrol creates these queues and configures them through out-of-band scripting commands.

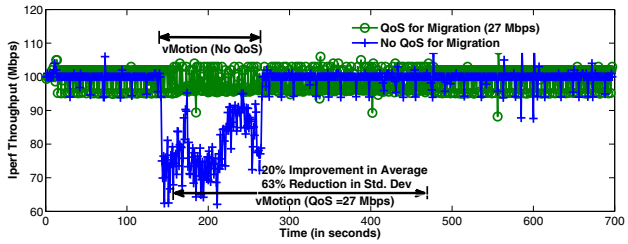
III. Experimental Evaluation

In this section, we evaluate the benefits of applying QoS in the form of bandwidth reservation for VM migration flows in presence of other flows. We use our cost of migration model to set the appropriate bandwidth for migrating a given VM (with a given VM size and a page dirty rate) such that it finishes within the specified time limit. Since these experiments require multiple hosts and switches, we conducted them in a virtual software testbed shown in Figure 1(c).

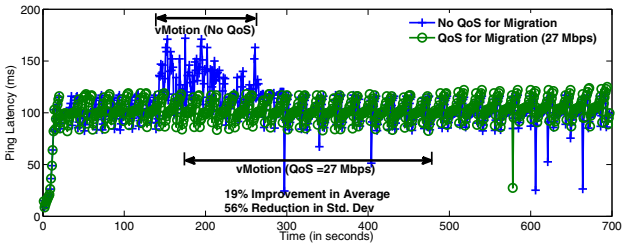
VM Workloads: On the migrating VM (with size 1024 MB) on host H1, we ran two workloads. First, we ran a page modification microbenchmark [7] at a given constant page dirty rate. Next, we deployed a 3 tier J2EE benchmark application called "RUBiS" [11]. RUBiS is an auction site prototype modeled after eBay.com that is used to evaluate application servers performance scalability. It has a 3 tier architecture with a client driver, an application server (we used JONAS), and a database (we used MySQL). RUBiS defines 26 interactions that can be performed such as browsing items by category or region, bidding, buying or selling items, leaving comments on other users and consulting ones own user page (known as myEbay on eBay). We used the bidding mix workload for our runs that includes 15% read-write interactions, while the rest are read-only browse interactions.

Topology Used: We created a small topology of 8 hosts, and 7 switches arranged in 3 layers inside the Mininet emulator [12] with multiple OpenFlow software switches running inside a VM. Simulated hosts (as well as OpenFlow switches) are created as processes in separate network namespaces inside the Mininet VM. Connectivity between hosts and switches, and between switches is obtained using virtual interfaces and each host, switch and the controller has a virtual interface (half of a "veth" pair). We connect two real hosts (in this case, the virtual ESX hosts) on two interfaces of the Mininet VM such that they are connected to the data center network created inside the Mininet VM. Out of the 8 hosts, 6 hosts are created inside the Mininet data center network (as processes that communicate over virtual ethernet links in their own network namespaces) and 2 hosts are external virtual ESX hosts with VMs that can be migrated over the shared data center network of 7 OpenFlow switches. In order to evaluate the effect of this VM migration on other flows in the network, we ran two other flows - TCP Iperf (bandwidth sensitive) and a ping (latency sensitive). The overall network topology implemented on our virtual testbed is shown in Figure 1(c). In all the experiments, a VM on host H1 is migrated from virtual ESX1 to virtual ESX2.

Constant page dirty rate experiment: We executed the page modification microbenchmark [7] at a constant page dirty rate of 50 pages per second on a VM on host H1 and migrated it from virtual ESX1 to virtual ESX2. In order to evaluate the effect of this migration on an Iperf flow and a ping flow in the network (refer Figure 1(c)), we performed this experiment twice - once without any QoS and second time after reserving a bandwidth of 27 Mbps (lower than what it would have got with



(a) Effect of VM migration on competing flows: Iperf throughput drops when there is no QoS for VM migration

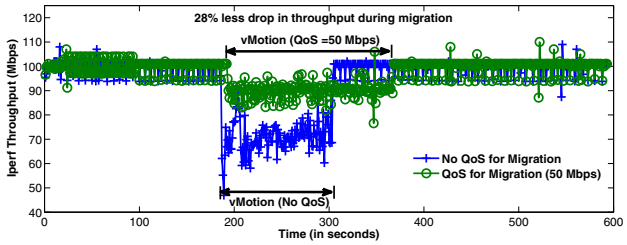


(b) Effect of VM migration on competing flows: Ping latency increases when there is no QoS for VM migration

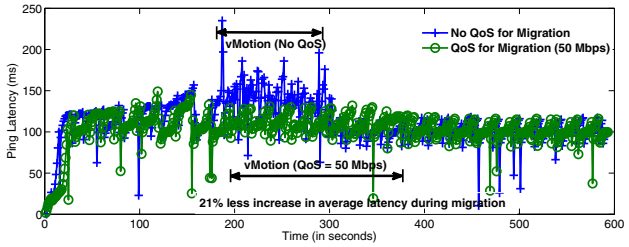
Fig. 2. Benefits of using QoS for VM migration at constant page dirty rate: Without QoS, Iperf throughput and ping latency (competing flows in the network) show a degradation of close to 20% in average performance and 55-65% in standard deviation during VM migration.

no QoS but sufficient to meet the migration deadline of 330 seconds at 50 pages/second dirty rate). The results are shown in Figure 2(a) for Iperf flow and in Figure 2(b) for ping. Without QoS, Iperf throughput and ping latency (competing flows in the network) show a performance degradation of close to 20% in average and 55-65% in standard deviation. With QoS (bandwidth reservation of 27 Mbps for VM migration) Iperf throughput and ping latency shows almost negligible degradation.

Variable page dirty rate experiment: Next, we executed a real benchmark application - RUBiS on the migrating VM on host H1 instead of the page modification benchmark at a constant page dirty rate. This VM was migrated from virtual ESX1 to virtual ESX2 as in the previous experiment, while rest of the network had an Iperf flow and a ping flow as shown in Figure 1(c). Since we do not have any explicit control over the page dirty rate behavior of RUBiS, we first had to monitor its page dirty rate at various loads, and use the observed average page dirty rate. We used 25 concurrent clients in RUBiS and the observed average page dirty rate for this load was around 400 pages/second, while the maximum rate was around 800 pages/second. We reserved a bandwidth of 50 Mbps as per maximum observed page dirty rate so that migration finishes within 330 seconds. Reserving bandwidth as per average page dirty rate resulted in VMware ESX hypervisor detecting lack of enough progress in "vMotion", since the page dirty rate can show a lot of variation for a real application. The results are shown in Figure 2(a) for Iperf flow and in Figure 2(b) for ping. With QoS (bandwidth reservation of 50 Mbps for VM migration) Iperf throughput and ping latency (competing flows in the network) show 20-27% less degradation in average performance and 40-60% less degradation in standard deviation. The VM migration time increased from 133 seconds to 194 seconds (45% increase).



(a) Effect of VM migration on competing flows: Iperf throughput drops when there is no QoS for VM migration



(b) Effect of VM migration on competing flows: Ping latency increases when there is no QoS for VM migration

Fig. 3. Benefits of using QoS for VM migration at variable page dirty rate (RUBiS): With QoS, Iperf throughput and ping latency (competing flows in the network) show 20-27% less degradation in average performance and 40-60% less degradation in standard deviation during RUBiS VM migration.

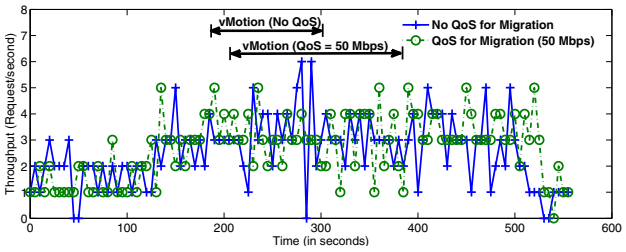
Note that, the migration time with QoS was 194 seconds (much lower than the specified deadline time limit of 330 seconds) which indicates the effective page dirty rate was around 250 pages/second.

The effect of reducing the link bandwidth for VM migration on the application running on the migrating VM (RUBiS AppServer) is shown in Figure 4(a) and Figure 4(b). Both throughput and latency remain largely unchanged. This illustrates that even though the migration time for the VM increased by 45%, it did not degrade the actual application performance on the migrating VM (at the loads we evaluated), while the interference to other flows (Iperf and ping) in the network was reduced as discussed above. Note that RUBiS latency reduces after the migration since the AppServer VM is migrated in a network-aware manner closer to the RUBiS DB VM and the RUBiS client and it no longer has any traffic on the bottleneck link (refer Figure 1(c)).

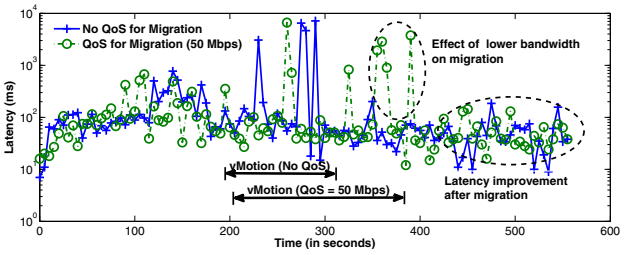
IV. Background and Related Work

A number of recent efforts [13] [14] [15] implement a QoS framework in which all flows are given some guarantees based on application specified SLA. In a real data center network, we envision that VMPatrol QoS controller will work with such existing techniques to provide QoS for all flows.

Kim et al. [13] propose a QoS framework that utilizes the global view of the network provided by OpenFlow, and present a technique to set low level QoS knobs based on high level descriptions of application requirements. Curtis et al. [14] describe a hybrid approach for traffic management in data centers where large flows (or elephant flows) are detected at the end hosts and the network controller is informed, which in turn finds the least congested path for such heavy flows using an increasing first fit algorithm. In a related work, the "Seawall" system presented by Shieh et al. in [16] provides a mechanism for data center network to apportion bandwidth



(a) RUBiS (migrating VM) throughput - remains unchanged



(b) RUBiS (migrating VM) latency - largely remains unchanged

Fig. 4. Application performance on the migrating VM (RUBiS AppServer VM): As bandwidth provisioned for migration is reduced, it does not degrade the actual application performance on the migrating VM, even though the migration time for the VM increases by 45%.

across various entities. Rajanna et al. [17] describe a system called XCo, where a central controller detects bottleneck links in the network and explicitly issues transmission directives to the end hosts to temporally separate transmissions competing for those bottleneck links. Civanlar et al. [15] describe an architecture to support QoS flows in an OpenFlow environment, where the controller receives a QoS contract from the sender (streaming server in their example), configures the forwarders for the QoS flow, monitors the network for an appropriate level of QoS performance, and initiates a switchover to an alternate route under congestion or network failure. In an earlier work, TCP Nice [18] was proposed as a mechanism to allow background TCP flows. A VM migration can not be considered a background flow (as proposed in TCP Nice) since it usually has an expected deadline and desired downtime.

V. Conclusion

In this paper, we presented the design and implementation of VMPatrol - a QoS framework for VM migrations. VMPatrol builds on a cost of VM migration model that is used to allocate a minimal bandwidth for a migration flow such that it completes within the specified time limit while causing minimal interference to other flows in the network. VMPatrol has been implemented in an OpenFlow based system for network-aware steady state VM management. Our experimental evaluation demonstrated that automated bandwidth reservation can reduce the impact of migrations on other flows in the network to a negligible level.

As part of ongoing and future work, we are working on extending VMPatrol's analytical model to work in presence of variable link bandwidths and variable page dirty rates. We also plan to work on extending the cost model to other hypervisors such as Xen and Hyper-V.

References

[1] "ESX Configuration Guide." [Online]. Available: http://www.vmware.com/pdf/vsphere4/r40/vsp_40_esx_server_config.pdf

[2] "Hyper-V: Live Migration Network Configuration Guide." [Online]. Available: http://technet.microsoft.com/en-us/library/ff428137(WS.10).aspx
[3] "VMware NetIOC (Network IO Control)." [Online]. Available: http://www.vmware.com/files/pdf/techpaper/VMW-Whats-New-vSphere41-Networking.pdf
[4] "KVM Migration," http://www.linux-kvm.org/page/Migration.
[5] "OpenFlow Slicing." [Online]. Available: http://www.openflow.org/wk/index.php/Slicing
[6] "Cisco UCS Quality of Service Configuration." [Online]. Available: http://www.cisco.com/en/US/products/ps10278/products_configuration_example09186a0080ae54ca.shtml
[7] 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 *IFIP Networking*, 2012.
[8] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines." in *NSDI*, 2005.
[9] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: Towards an Operating System for Networks," in *ACM SIGCOMM CCR*, July 2008.
[10] "VMWare vSphere command line interface," http://www.vmware.com/support/developer/vcli/.
[11] "RUBiS Benchmark," http://rubis.ow2.org/.
[12] "Mininet: rapid prototyping for software defined networks," http://yuba.stanford.edu/foswiki/bin/view/OpenFlow/Mininet.
[13] W. Kim, P. Sharma, J. Lee, S. Banerjee, J. Tourrilhes, S. Lee, and P. Yalagandula, "Automated and scalable qos control for network convergence," in *ACM INM/WREN*, 2010.
[14] A. Curtis, K. Wonho, and P. Yalagandula, "Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection," in *IEEE INFOCOM*, 2011.
[15] S. Civanlar, M. Parlakisik, A. Tekalp, B. Gorkemli, B. Kaytaz, and E. Onem, "A qos-enabled openflow environment for scalable video streaming," in *IEEE GLOBECOM Workshops*, 2010.
[16] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha, "Sharing the data center network," in *USENIX NSDI*, 2011.
[17] V. Rajanna, S. Shah, A. Jahagirdar, C. Lemoine, and K. Gopalan, "Xco: explicit coordination to prevent network fabric congestion in cloud computing cluster platforms," in *ACM HPDC*, 2010.
[18] A. Venkataramani, R. Kokku, and M. Dahlin, "Tcp nice: A mechanism for background transfers." in *OSDI*, 2002.