Remedy: Network-Aware Steady State VM Management for Data Centers

Vijay Mann, Akanksha Gupta, Partha Dutta, Anilkumar Vishnoi, Parantapa Bhattacharya, Rishabh Poddar, Aakash Iyer

Remedy: Network-aware Steady State VM Management for Data Centers

Vijay Mann[1], Akanksha Gupta[1], Partha Dutta[1], Anilkumar Vishnoi[1], Parantapa Bhattacharya[2], Rishabh Poddar[2], and Aakash Iyer[1]

[1] IBM Research, India
{vijamann, avishnoi, akguptal, parthdut, aakiyer1}@in.ibm.com
[2] Indian Institute of Technology, Kharagpur, India
parantapa@cse.iitkgp.ernet.in,rishabh.pdr@gmail.com

Abstract. Steady state VM management in data centers should be network-aware so that VM migrations do not degrade network performance of other flows in the network, and if required, a VM migration can be intelligently orchestrated to decongest a network hotspot. Recent research in network-aware management of VMs has focused mainly on an optimal network-aware initial placement of VMs and has largely ignored steady state management. In this context, we present the design and implementation of Remedy. Remedy ranks target hosts for a VM migration based on the associated cost of migration, available bandwidth for migration and the network bandwidth balance achieved by a migration. It models the cost of migration in terms of additional network traffic generated during migration.

We have implemented Remedy as an OpenFlow controller application that detects the most congested links in the network and migrates a set of VMs in a network-aware manner to decongest these links. Our choice of target hosts ensures that neither the migration traffic nor the flows that get rerouted as a result of migration cause congestion in any part of the network. We validate our cost of migration model on a virtual software testbed using real VM migrations. Our simulation results using real data center traffic data demonstrate that selective network aware VM migrations can help reduce unsatisfied bandwidth by up to 80-100%.

Keywords: network aware VM migration; VM migration traffic modeling
Corresponding author: Vijay Mann, IBM Research, 4, Block C, Institutional Area, Vasant Kunj, New Delhi - 110070, INDIA. Email: vijamann@in.ibm.com.

1 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 of the existing work on virtual machine management has focused on CPU and memory as the primary resource bottlenecks while optimizing VM placement for server consolidation, dynamic workload balance, or disaster recovery [2,10,16]. They do not take into account network

topology and current network traffic. However, steady state management of VMs should be network-aware so that migrations do not degrade network performance of other flows in the network, and if required, a VM migration can be intelligently orchestrated to decongest a network hotspot. If the trigger to migrate one or more VMs comes from the network itself, it should ensure that the number of VM migrations are kept low. Recent research [11] [7] [14] [13] in network-aware VM management has focused mainly on an optimal network-aware initial placement of VMs and has largely ignored steady state management.

In this context, we present the design and implementation of Remedy: a system for network-aware steady state management of VMs. To the best of our knowledge, Remedy is the first system to perform network-aware management of VMs so as to minimize the cost of VM migration over a long-term. We make the following contributions in this paper:

- We model the cost of VM migration in terms of total traffic generated due to migration.
- We present a target selection heuristic (TSH) that ranks target hosts for a VM migration based on the associated cost of migration, available bandwidth for migration and the network bandwidth balance achieved by a migration.
- We implement Remedy as an OpenFlow [12] controller application that can detect the most congested links in a data center network, identify the flows that pass through those links, and select the best flows to be rerouted that can resolve network congestion. [3] It then uses an intelligent VM selector heuristic (VSH) to select VMs that are part of these flows based on their relative cost of migration and then migrates these VMs in a network-aware manner to decongest these links. Our target selection heuristic (TSH) ensures that neither the migration traffic nor the flows that get rerouted as a result of migration cause congestion in any part of the network.
- We also implement our prototype in the VMFlow simulation framework [11] from our earlier work. Our simulation results using real traffic data from a data center demonstrate that selective network aware VM migrations can help reduce unsatisfied bandwidth by up to 80-100%.
- We evaluate the Remedy OpenFlow prototype on a virtual software testbed. Our results validate our cost of migration model.

The rest of this paper is organized as follows. Section 2 presents a description of the problem and describes our cost of migration model. Section 3 describes the design and architecture of Remedy. We provide an experimental evaluation of various aspects of Remedy in Section 4. An overview of related research is presented in Section 5. Finally, we conclude the paper in Section 6.

2 Online Minimum Cost Network Utilization using VM Migration

In this section we formulate the main optimization problem that need to be solved for network-aware steady state management of VMs. Since the arrival, departure and change of network demands is not known in advance, the problem is an online one.

[3] Remedy can possibly be implemented over other network monitoring and control frameworks. We chose OpenFlow for ease of implementation.

Context: The data center network is modeled as a directed graph $G(V, E)$, where each link e in E has a capacity $C(e)$. There are two types of vertices in V: the hosts and the network switches. The edges represent the communication links. Each host has a processing capacity.[4] A network demand is defined by the three parameters, its source, destination and its rate.

Configuration: At any given point of time, the configuration of the system consists of the set of VMs (with their current processing requirement), the set of existing demands (with their current rate), the *placement* Π that maps VM to hosts, and the routing RT that maps demands to paths (in G). (A flow consists of a demand along with its routing path.) The routing is unsplittable, i.e., all the traffic for a demand is sent using a single flow over a single path. Note that multiple VMs can be mapped to a single host.

The load of a link is the sum of the rates of all flows that are routed over that link. The utilization (also called congestion) of a link is defined as the ratio of the load on the link to its capacity. We say that network has α*-utilization* in a configuration, if the utilization of every link is at most α.

Constraints: A configuration should satisfy the capacities of all the hosts and the links. In other words, the sum of the processing requirements of all VMs that are placed on a host should be less than or equal to the capacity of the host, and the sum of the rates of all flows routed over a link should be less than or equal to the capacity the link.

Migration Cost: Migrating a VM generates network traffic between the source and the destination hosts of the migration, where the amount of traffic depends on the VMs image size, its page dirty rate, the migration completion deadline and the available bandwidth. We model the cost of a migration as the total traffic generated due to the migration. In this paper, we model the migration cost of VMware vMotion [3]. Note that, even though the model presented in this paper is for VMware vMotion, our techniques are generic enough to be applicable for most live migration systems that are based on iterative pre-copy technique. In particular, the model presented in this work can be extended by making minor modifications to the stop conditions that vary across different hypervisors.

VMware vMotion[5] is a pre-copy live migration technique. The conditions for stopping the pre-copy cycle are as follows:

1. The current dirty memory can be transmitted in T milliseconds (i.e., dirty memory is small enough to start a stop-copy of at most T millisecond). Here, T is a user setting, and it is called the switchover goal time.
2. vMotion did not make *enough progress* in the previous pre-copy cycle. vMotion measures the progress of a migration by the difference in dirty memory before and after a pre-copy cycle. This difference should at least be X MBs, where X is a user setting. We call X the minimum required progress amount.

In case vMotion did not make enough progress, and the current dirty memory cannot be transmitted in 100 seconds, the migration fails. In the following Theorem, we

[4] Although, storage and memory requirements/capacities of the VMs and the hosts are not considered in this paper, our solution can be extended for these requirements.

[5] For ease of presentation, we use the terms vMotion and VM migration interchangeably.

derive 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: (1) the memory size M of a VM, (2) the page dirty rate R of a VM and (3) the bandwidth of the link used for migration L.

Theorem 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* $N = M \cdot \frac{1-(R/L)^{n+1}}{1-(R/L)}$.

Proof. Let N_i denote the traffic on the i^{th} pre-copy cycle. Taking into account the page dirty rate R and the bandwidth L used for migration, we see that the first iteration will result in migration traffic equal to the entire memory size $N_0 = M$, and time $\frac{M}{L}$. During that time, $M\frac{R}{L}$ amount of memory becomes dirty, and hence, the second iteration results in $N_2 = M\frac{R}{L}$ amount of traffic. Extending this argument it is easy to see that $N_i = M\left(\frac{R}{L}\right)^{i-1}$ for $i \geq 1$. Thus if there are n pre-copy cycles and one final stop-copy cycle, the total traffic generated by the migration is $N = \sum_{i=1}^{n+1} N_i = M \cdot \frac{1-(R/L)^{n+1}}{1-(R/L)}$. We now derive an equation for n.

Suppose that the pre-copy stops after n cycles. Then one of the following two conditions must hold true, at the end of cycle n. (1) From the first pre-copy stopping condition we have, $M\left(\frac{R}{L}\right)^n < T \cdot L$, which in turn implies $n < \log_{R/L} \frac{T \cdot L}{M}$. (2) From the second pre-copy stopping condition we have, $M\left(\frac{R}{L}\right)^{n-1} - M\left(\frac{R}{L}\right)^n < X$, which in turn implies $n < \log_{R/L} \frac{X \cdot R}{M \cdot (L-R)}$.

Thus, from the above two conditions, the number of pre-copy cycles $n = \min\left(\left\lceil \log_{R/L} \frac{T \cdot L}{M} \right\rceil, \left\lceil \log_{R/L} \frac{X \cdot R}{M \cdot (L-R)} \right\rceil\right)$.

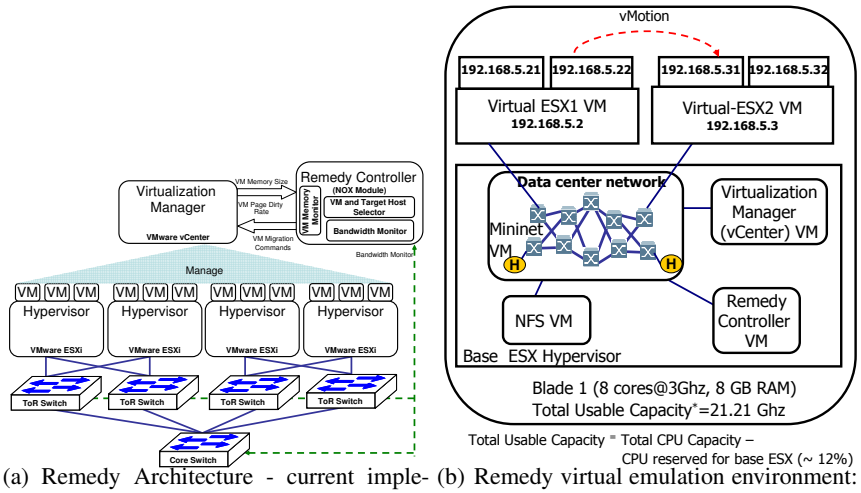
We also note that the time $W(L)$ spent on the stop-copy transfer for a given migration bandwidth L, is given by $W(L) = \frac{M}{L} \cdot \left(\frac{R}{L}\right)^n$. ■

We now describe the online problem that we address to perform network-aware steady state VM management.

Problem: Due to possibly under-provisioned links and time-varying demands, the network utilization may exceed a desired threshold over time. In this problem, we aim to reduce the network utilization by migrating VMs while incurring minimum migration cost. More specifically, we define the online α-Minimum Migration Cost Network Utilization (α-MCNU) problem as follows.

During the data center operation, if there is a change in the demands (in particular, if some demand is created, terminated, or its rate is changed significantly), then the system may introduce one or more VM migrations. Each migration has an upper bound on its completion time. In the α-MCNU problem, given an initial configuration (placement and routing), and the arrival, departure and change of demands over time (which is not known in advance), we need to select a sequence of VM migrations such that the network utilization remains at most α, except during the period of a migration, while ensuring that the cost of migration is minimized. Even the offline version of MCNU problem, where all demand changes are known in advance, is NP-Hard. [6]

[6] We omit the proof due to lack of space



(a) Remedy Architecture - current implementation with VMware and NOX OpenFlow controller

(b) Remedy virtual emulation environment: emulates 2 physical hosts with 4 mobile VMs using nested virtualization

Fig. 1. Remedy Architecture and Virtual Emulation Environment

Theorem 2. *The offline version of MCNU problem is NP-Hard.*

We present our heuristics for the MCNU problem in section 3.2.

3 Architecture and Design

In this section we present the architecture and design of Remedy, a system for network-aware steady state VM management (refer Figure 1(a)). Remedy has been implemented in Python as a NOX [9] module, which is one of the popular OpenFlow controllers currently available. An OpenFlow controller is connected to all OpenFlow enabled switches and routers and communicates with these switches through a control channel using the OpenFlow control protocol. Remedy has three main components: (1) Bandwidth Monitor, (2) VM and Target Selector and (3) VM Memory Monitor. We now describe each of the above components in detail in the following subsections.

3.1 Bandwidth Monitor

OpenFlow provides flow monitoring capabilities in the form of per flow, per port and per queue statistics (using OFPST_FLOW, OFPST_PORT and OFPST_QUEUE request types, respectively as per OpenFlow 1.1 specification). Remedy Bandwidth Monitor first detects all links by using the "discovery" module in NOX. Each link is represented as a 4 tuple: `<Start SwitchID,Start PortNumber,End SwitchID,End PortNumber>`.

Remedy controller polls for flows statistics and port statistics by periodically sending flowStats (OFPST_FLOW) and portStats (OFPST_PORT) requests to all the switches in the network. It uses the port statistics to find out the transmitted bytes from each port on each switch in the current polling interval and sorts this list in decreasing order to find the most busy ports in the network. For each port in the busy ports list, it then uses the flow statistics to find out the number of bytes recorded for each flow on that

```
1: function VSH
2:     rank VMs in increasing order of their approx migration cost = 120% of VM size
3:     rank VMs in the increasing order of their number of communicating neighbor
4:     rank VMs in the increasing order of their total input/output traffic
5:     return VM with lowest sum of the three ranks
6: function TSH (VM v)
7:     H ← set of all hosts with enough spare CPU capacity to place v; H_f ← ∅
8:     calculate available b/w from current host of v to each host in H
9:     for each host h in H do
10:        calculate the migration traffic and migration feasibility of VM v to host h following Sec. 2
11:        if migrating v to h is feasible then
12:            add h to H_f
13:            calculate available b/w for each flow of v when re-route to host h
14:            calculate normalized weighted sum of available b/ws calculated in previous step
15:    return a ranked list of hosts in H_f in the decreasing order of their normalized weighted sum
16:
```

Fig. 2. Pseudocode for VSH and TSH

port in the current monitoring interval. Note that a flow represents a network demand and is uniquely identified through source and destination IP addresses (we ignore port numbers since our eventual goal is to find VMs that contribute the most to a link's utilization). This list of flows on each port is also sorted in decreasing order to find the heaviest flows on each port in the network. Since a port may appear on either end of a link, the list of busy ports is used to calculate the traffic on each link and a list of busy links is created. Similarly, the list of heaviest flows on a port is used to calculate the heaviest flows on each link, and their contribution to the overall link utilization is calculated.

Remedy uses an input headroom value to determine a link utilization value beyond which a link is considered congested. For example, if the input headroom value is 20, then a link is considered congested when its utilization crosses 80%. Since a VM may be part of multiple flows either as a source or a destination, the sorted list of heaviest flows on each congested link is used to determine the VMs which are top contributors to a link's utilization. This list of VMs is passed on to the VM and Target Host Selector.

3.2 VM and Target Host Selector

The VM and Target Host Selector uses two methods, the VM selector heuristic (VSH) and the target selector heuristic (TSH), to select a VM and the target host for migrating the selected VM, in order to control network utilization. These two heuristics together addresses the MCNU problem with $\alpha = 1$ (as defined in Section 2). In the case where the VM to be migrated is already provided as input (either by a user or by an existing VM placement engines which consider CPU and memory statistics to migrate a VM for server consolidation, workload balance or fail-over), Remedy uses only the TSH. We now describe these two heuristics in detail. (See Figure 2 for pseudocode.)

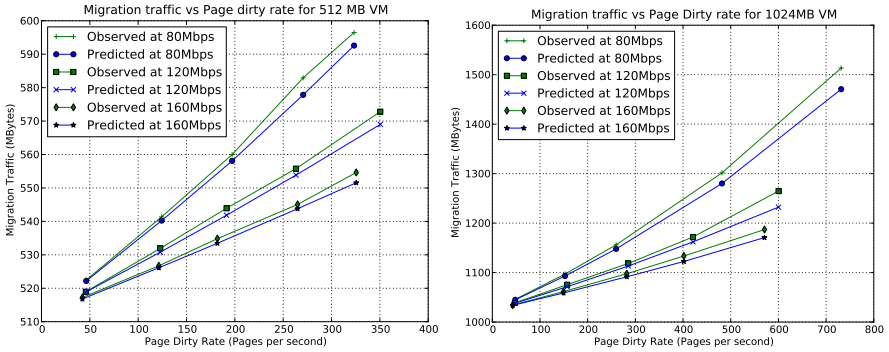
VM Selector Heuristic (VSH): VM selector heuristic uses the list of top contributor VMs for each congested edge and selects the VMs to be migrated based on number of communicating neighbors of a VM, total input/output traffic of a VM and an approximate cost of migrating a VM. As discussed in section 2, the cost of migrating a VM (in terms of additional network traffic generated) depends on the memory size of the VM, its page dirty rate, the available bandwidth for migration, and some other hypervisor specific constants. Since, the available bandwidth for migration can be known

only when the target host for a migration has already been determined, we use 120% of memory size of a VM as a rough indicator of the additional network traffic that would be generated on migrating a particular VM. VMs are first sorted in increasing order by this approximate cost of migration and then by their number of communicating neighbors (in increasing order) and finally by their total input/output traffic. The rank of each VM in these three sorted lists is noted and added together. The VM which has the lowest sum of ranks is selected as the VM to be migrated.

Target Selector Heuristic (TSH): Given a VM to migrate, target selector heuristic ranks target hosts for that VM based on the associated cost of migration, available bandwidth for migration and the network bandwidth balance achieved by a migration. In a typical data center, a physical host has many VMs running on it, which are connected to physical network interfaces through a virtual switch (or vSwitch) inside the hypervisor. Physical hosts are directly connected to a ToR switch, which in turn is connected to one or more aggregate switches. The aggregate switches then connect to a layer of core switches. In some data centers, the aggregate and core switches may form a single layer. TSH evaluates all end hosts that have free CPU capacity. For all such hosts, it finds the path from the current host and evaluates the available bandwidth. Note that, Remedy directly calculates the available bandwidth on a path using the flow statistics collected by Bandwidth Monitor from all switches. Thus, Remedy does not need to use any available bandwidth estimation tools. Based on the observed bandwidth, Remedy calculates the cost of migration as per the cost of migration model described in Section 2 (which is in terms of additional network traffic that will be generated during migration). This additional network traffic is added to existing traffic on the chosen path and the resultant available bandwidth is calculated to ensure that migration is feasible. If migration is feasible, TSH then calculates the available bandwidth for all network demands or flows that this VM is a part of and which will get rerouted as a result of this migration. TSH calculates a normalized weighted sum of these available bandwidths (each demand from a VM is first normalized with respect to the largest demand for that VM, and then multiplied by the available bandwidth on its path - this step is repeated for all demands for a given VM, and these are added up across all network demands of that VM). This normalized weighted sum of available bandwidths is used to create a ranked list of all destination hosts to which a migration is feasible. Intuitively, this normalized weighted sum of available bandwidths favors large flows over small flows and the heuristic attempts to create the best possible available bandwidth balance after migration. The Remedy System tries to migrate the selected VM to the destination host one by one, according to the above ranked list, until a migration is successful.

3.3 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 ESXi hypervisor as our virtualization platform that support live VM migration in the form of VMware “vMotion”. VMware ESXi hypervisor has a remote command line interface (RCLI) [4], 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 ESXi logs to find out page dirty rates.



(a) VM size = 512 MB (b) VM size = 1024 MB

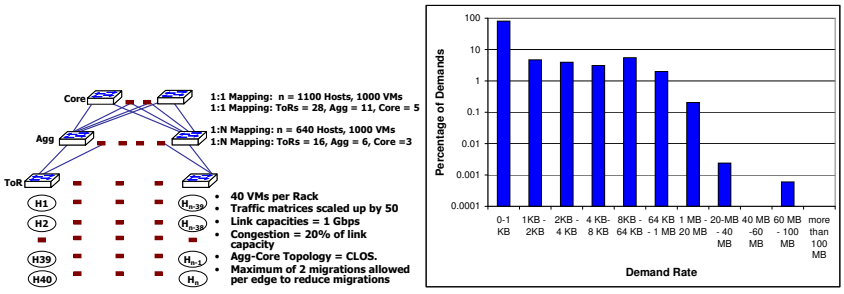
Fig. 3. Cost of migration model predicts migration traffic to within 97% accuracy

4 Experimental Evaluation

We first validated the correctness of our system on an innovative virtual software environment that leverages nested system virtualization and software switches. In order to analyze the effects of VM selector and target selector heuristics at scale, we implemented these heuristics in the VMFlow simulation framework from our earlier work [11] and conducted simulations using traffic data from a real data center. We now describe these experiments in detail.

4.1 Validating Remedy in an OpenFlow Network

Virtual Software Testbed: We created an innovative virtual emulation environment that gave us full control over the network as well as VM mobility across hosts. We leveraged nested server virtualization in VMware ESXi server [5] to create VMs that run hypervisor software within a single physical machine. Our virtual testbed that emulates two hosts (each with two VMs) inside a single physical blade server is shown in Figure 1(b). The various VMs that run the hypervisor software (we refer to them as virtual hypervisors) represent multiple virtual hosts. Virtual hypervisor or virtual hosts are connected to each other through an OpenFlow software switch (version 1.1) or a Mininet [1] data center network with multiple OpenFlow software switches running inside a VM. Mininet uses process-based virtualization and network namespaces to rapidly prototype OpenFlow networks. Simulated hosts (as well as OpenFlow switches) are created as processes in separate network namespaces. 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 ESXi hosts) on two interfaces of the Mininet VM such that they are connected to the data center network created inside the Mininet VM. We run the Remedy controller on top of NOX as a remote controller inside another VM and this is also connected to the Mininet data center network. Shared storage for these virtual hosts is provided by shared NFS service also running inside a VM and connected to the Mininet data center network. Virtualization manager (VMware vCenter) runs inside another VM. Direct wire connectivity between these various VMs (OpenFlow or Mininet VM, Controller VM, NFS VM, vCenter VM and Virtual Hypervisor VMs) is achieved through various vSwitches on the base hypervisor. We also created VMs on top of the virtual hypervisors (or virtual hosts) that can be migrated across various hosts.



(a) Topology used (b) Distribution of network demands

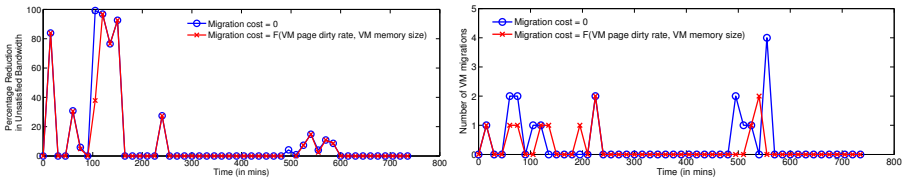
Fig. 4. VMFlow simulations to evaluate the VM selector and Target selector heuristics

While, the virtual testbed provides only an emulation environment and can not be used for any scalability or absolute performance comparison, it still proves to be a valuable tool to verify the correctness of our system using real VM migrations on VMware ESXi hypervisor, and OpenFlow capable software switches.

Results: In order to validate our cost of migration model described in section 2, we developed a simple page modification microbenchmark. This microbenchmark, first allocates large chunks of page aligned memory. It then randomly writes to different pages in a loop and executes busy loop for a predetermined count based on the specified page dirty rate. The page modification benchmark is executed with different page dirty rates on one of the VMs hosted on the virtual ESXi server. The two virtual ESXi servers were connected via an OpenFlow switch VM for this experiment. This VM was then migrated to another virtual ESXi at different link bandwidths. To create different link bandwidths, a queue was created on the output port of the OpenFlow switch and the minimum rate on that queue was set to the desired value. Remedy controller installed a rule to enqueue all migration traffic (identified through TCP port numbers used by "vMotion") to this queue and all other traffic was sent through the default queue. The total migration traffic was monitored inside the Remedy Controller, as well as verified through VMware ESXi logs. The page dirty rates were also verified through VMware ESXi logs. The results are shown in Figure 3 for two VM sizes - 512 MB and 1024 MB. In both the cases, our model predicts the total migration traffic to within 97% accuracy for all page dirty rates and link bandwidths.

4.2 Validating the effectiveness of VM and target selector heuristics at scale through simulations

In this section we evaluate the VM selector and target selector heuristics at scale through simulations in the VMFlow simulation framework [11]. Note that in all these simulations, the cost of migration was assumed to be 1.2 times the memory size of a VM as mentioned in section 3. Deadline for migration completion time was assumed to be equal to the monitoring interval for our input traffic matrices (15 minutes). The memory sizes for all VMs were assumed to be normally distributed with a mean of 1 GB and standard deviation of 512 MB. With these settings, the goal of the simulation experiments is to evaluate the effectiveness of VM and target selector heuristics at scale in achieving network balance and their ability to satisfy bandwidth demand.



(a) Reduction in unsatisfied bandwidth due to network-aware steady state VM management

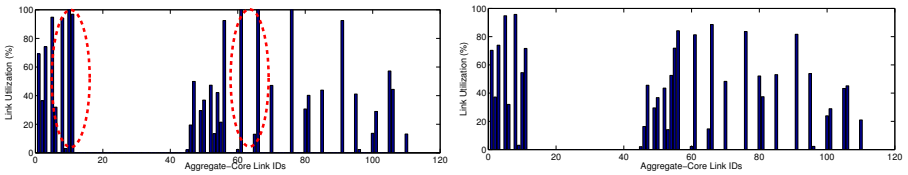
(b) Improvement in number of migrations due to migration cost-awareness

Fig. 5. Results from VMFlow simulator with 1000 VMs and 1:1 host to VM mapping using data from a real data center over a 12 hour period: Results illustrate benefits of using network-aware steady state VM management - unsatisfied bandwidth reduces up to 100% at some time-steps in the simulation, adding migration cost-awareness to the heuristics reduce the total number of migrations required to achieve this benefit from 17 to 11

VMFlow Simulator: VMFlow simulator simulates a network topology with VMs, switches and links as a discrete event simulation. Each host is assumed to contain one or more VMs. VMs are either randomly mapped to all hosts (in case of 1:1 host to VM mapping) or they are mapped based on their CPU requirements (in case 1:N host to VM mapping). VMs run applications that generate network traffic to other VMs, and VMs can migrate from one node to the other. At each time step, network traffic generated by VMs (denoted by an entry in the input traffic matrix) is read and the simulator attempts to satisfy these demands in decreasing order (largest demands are satisfied first). For a given demand, it calculates the corresponding route between the hosts based on an already used path or the shortest path in the network topology, while ensuring that available link capacities can satisfy the given network demand. In order to simulate congestion, we modified the simulator so that it can take more demands than available link capacities. For the results in this paper, we simulated 20% congestion where each link can take 20% more traffic than its rated capacity. The simulator then executes an ongoing optimization step in which it detects congestion, finds the set of VMs to migrate to decongest the network using VM selector heuristic (VSH) and selects the appropriate set of destinations for each VM using the target selector heuristic (TSH). At each time step of the simulation, we compute the total bandwidth demand that can not be satisfied, the number of migrations required and link utilizations. A new input traffic matrix that represents the network traffic at that time instance, is used at each time step of the simulation.

Network Topology and Input Traffic Matrices: The simulator creates a network based on the given topology. Our network topology consisted of 3 layers of switches: the top-of-the-rack (ToR) switches which connect to a layer of aggregate switches which in turn connect to the core switches. Each ToR has 40 hosts connected to it. We assume a total of 1000 VMs and 1100 hosts for the 1:1 VM to host mapping scenario, and 640 hosts for 1:N host to VM mapping scenario. The topology used for these simulations is shown in Figure 4(a).

To drive the simulator, we needed a traffic matrix (i.e. which VM sends how much data to which VM). We obtained data from a real data center with 17,000 VMs with aggregate incoming and outgoing network traffic from each VM. The data had snapshots of aggregate network traffic over 15 minute intervals spread over a duration of 5 days. Our first task was to calculate a traffic matrix out of this aggregate per VM data. Given



(a) With no migration (b) After 1 network-aware migration

Fig. 6. Agg-Core layer link utilizations from VMFlow simulator with 1000 VMs and 1:1 host to VM mapping using data from a real data center at time step 2 (point 2 in Figure 5(a)). Results illustrate benefits of a single network-aware VM migration - links that were congested and hence could not take on additional bandwidth demands (in the no migration case) get decongested through 1 network-aware VM migration

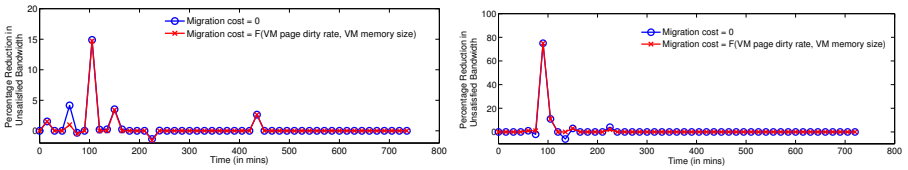
the huge data size, we chose data for a single day. We used the simple gravity model to create a traffic matrix out of this aggregate per VM data (as described in [11]). As noted in [11], these traffic matrices did not show much variation over time. Note that traffic matrices generated by simple gravity model tend to be dense and traffic is distributed over all the VMs in some proportion. This results in a large number of very low network demands. In order to make these demands significant so that they can cause congestion in the network, we used a scale-up factor of 50 (i.e. each entry in the input traffic matrix was multiplied by 50) for the traffic matrices used in 1:1 host to VM mapping experiment and scale up factors of 75 and 125 for the traffic matrices used in 1:N host to VM mapping experiments. A distribution of network demands for one of the traffic matrices (used for one time step in simulation) after applying a scale factor of 50 is shown in Figure 4(b). This matrix had 1 million total demands out of which only 20% are zero demands. Maximum demand size was 65.91 MB/s and the mean demand size was 13 KB/s.

1:1 host to VM mapping simulation results:

We first ran the simulation for 1:1 host to VM mapping. Since, the input traffic matrices are already dense (much more than a real data center traffic matrix), 1:1 host to VM mapping is useful to show the benefits of our heuristics. We conducted two sets of simulations:

- The first set of simulations, assumed that VM migrations were of 0 cost and the only objective was to decongest a given edge (with the constraint that only two migrations were allowed per congested edge).
- During our second set of simulations, we assigned a cost of VM migration based on the cost of migration model given in section 2 that uses memory size of a VM, its page dirty rate, and link bandwidth to predict the additional traffic generated on the network.

Simulation results for 1:1 host to VM mapping are shown in Figure 5(a) and in Figure 5(b). Results illustrate benefits of using network-aware steady state VM management. Unsatisfied bandwidth reduces up to 100% at some time-steps in the simulation (refer Figure 5(a)) while adding migration cost-awareness to the heuristics reduces the total number of migrations required to achieve this benefit from 17 to 11 (refer Figure 5(b). Link utilizations for the aggregate-core layer for one of the time steps (time step 2 in Figure 5) are given in Figure 6(a) for the no migration base case, and in Figure 6(b)



(a) Dense communication matrix (b) Sparse communication matrix

Fig. 7. Results from VMFlow simulator with 1000 VMs and 640 hosts (1:N host to VM mapping, 1000 VMs were mapped to 574 hosts using CPU data) using data from a real data center over a 12 hour period: Reduction in unsatisfied bandwidth is lower (up to 15%) for dense communication matrix generated by simple gravity model and goes up to 75% for a sparse matrix.

for the 0 cost network aware migration case. Note that the links that were congested (marked by circles) and hence could not take on additional bandwidth demands (in the no migration case) get decongested through a single network-aware VM migration.

1:N host to VM mapping simulation results:

In this section, we present the simulation results for 1:N host to VM mapping scenario. We assumed a fixed host CPU capacity of 4 GHz and generated normally distributed random CPU demands for each VM with a mean demand of 1.6 GHz and standard deviation of 0.6 GHz.

VMs are mapped based on their CPU requirements. 1000 VMs were mapped onto 574 hosts and the remaining hosts remained idle. While, choosing the best host for a VM under a given ToR switch, the host with the maximum available link bandwidth to the ToR switch is chosen, provided that the host has enough spare CPU capacity to host the VM. Traffic matrices were scaled up by a factor of 75.

In this case again we conducted two sets of simulations - one which assumed a 0 migration cost, and the other that calculated a cost of migration in terms of additional network traffic generated as per our cost of migration model. Simulation results for 1:N host to VM mapping are shown in Figure 7(a). Unsatisfied bandwidth reduces by up to 15% at some time steps in the simulation. However, the gains here are modest, as compared to 1:1 mapping scenario, since the traffic matrix is already dense and as more VMs are packed into a single host, the network traffic matrix becomes denser, reducing the ability to move VMs freely. However, real data center traffic matrices are much less dense (as compared to the traffic matrices generated by simple gravity model that we used), and hence we expect the gains to be much higher in real data centers. To verify this, we generated sparse matrices by randomly making half of the demands zero, while increasing the scale factor for traffic matrices from 75 to 125. Simulation results for the sparse matrices are given in Figure 7(b). Unsatisfied bandwidth now reduces by up to 75%. Adding migration cost-awareness to the heuristics reduces the total number of migrations required in both the scenarios (from 11 to 8 in the dense communication matrix, and from 15 to 13 in the sparse communication matrix scenario). Note that at one time step in dense communication matrices, the unsatisfied bandwidth increases resulting in negative reduction value. We believe this also happened primarily due to a dense traffic matrix, where the added congestion in the network resulted in the simulator trying to satisfy larger demands and failing, when it could have satisfied smaller demands.

5 Background and Related Work

Recently there has been significant amount of work on VM placement and migration. The related research work can be broadly classified as follows.

5.1 Network-aware placement of VMs

Our prior work on network-aware initial placement of VMs - VMFlow [11], proposed a greedy heuristic for initial placement of VMs to save network power while satisfying more network demands. As one of the contributions of this paper, VMFlow simulator has been enhanced with the VM selector heuristic (VSH) and TOR selector heuristic (TSH) to incorporate network-aware steady state management.

The online minimum congestion mapping problem studied in [7] by Bansal et al. is close to our work: it considers placing a workload consisting of network traffic demands and processing demands on an underlying network graph (substrate) while trying to minimize the utilization over all links and nodes. The paper [7], however, differs from our work in two crucial aspects. First, in [7], once a workload is placed (i.e., its node and link assignment are finalized by the algorithm), this assignment cannot be changed during placement of subsequent workloads. On the other hand, migrating the source and destination of a network demand (which corresponds to VMs in our setting) is the primary technique used in this paper. Second, we consider the cost of VM migration, and ensure that the migration is completed in a timely manner, which are not considered in [7]. Sonnek et al. [14] propose a technique that monitors network affinity between pairs of VMs and uses a distributed bartering algorithm coupled with migration to dynamically adjust VM placement such that communication overhead is minimized. Meng et al. [13] consider heuristics for VM placement in a data center to optimize network traffic while minimizing the cost of VM migration. Although their primary algorithm is presented in an offline setting, it can be executed periodically to modify the VM placements according to the time-varying network demands. However, unlike our paper, their placement algorithm does not consider minimizing the cost over a sequence of network demand changes, but only tries to minimize the cost of each individual step. Moreover, the solution presented by [13] does not specify the routing path for the traffic demands, which is handled by our work. Finally, compared to [13], our solution uses a more detailed model for the cost and time required for the VM migrations.

5.2 Cost-aware migration of VMs and prediction of migration performance

Most of the existing research [10] [8] [6] on cost-aware migration of VM analyzes the impact of a live VM migration on the workload that is executing on the migrating VM itself. Breitgand et al. [8] model the total cost of live in-band migration, and present an optimal bandwidth allocation algorithm for in-band live migration such that the cost of migrating a VM, in terms of the down time experienced by the migrating VM, is minimum. However, in a shared data center network, a VM migration also affects the performance of other hosts in the network, which is the focus of our work. Jung et al. [10] present a holistic optimization system that considers various costs of migration, such as, application performance degradation of the migrating VM and power trade offs. Sherif et al. [6] present two simulation models that are able to predict migration time to within 90% accuracy for both synthetic and real-world benchmarks running on Xen hypervisor. Stage et al. [15] propose a complementary scheme in which network topology aware scheduling models are used for scheduling live VM migrations such that effect of VM migrations on other nodes in the network is reduced.

6 Conclusion

In this paper, we presented the design and implementation of Remedy - a system for network aware steady state VM management that ranks target hosts for a VM migration based on the associated cost of migration, available bandwidth for migration and the network bandwidth balance achieved by a migration. Our simulation results using real data center traffic data demonstrated that selective network aware VM migrations, as proposed in Remedy, can help reduce unsatisfied bandwidth by up to 80-100%.

As part of ongoing and future work, we are currently working on implementing Remedy in a real data center with state-of-the-art network equipment. Future extensions to Remedy include a decentralized design for monitoring and an analysis engine to analyze network demands and link utilizations using auto regressive moving average model to verify whether a given demand or link utilization is stable or a transient spike. This will help us make more informed decisions to tackle long term congestion. Finally, we also plan to work on extending the cost of migration model to other hypervisors such as Xen and KVM.

References

1. Mininet: Rapid prototyping for software defined networks, http://yuba.stanford.edu/foswiki/bin/view/OpenFlow/Mininet
2. VMware Server Consolidation, http://www.vmware.com/solutions/consolidation/
3. VMware vMotion: Live migration of virtual machines, http://www.vmware.com/products/vmotion/overview.html
4. VMWare vSphere command line interface, http://www.vmware.com/support/developer/vcli/
5. VMware vSphere Hypervisor, http://www.vmware.com/products/vsphere-hypervisor/overview.html
6. Akoush, S., Sohan, R., Rice, A., Moore, A., Hopper, A.: Predicting the performance of virtual machine migration. In: IEEE MASCOTS (2010)
7. Bansal, N., Lee, K., Nagarajan, V., Zafer, M.: Minimum congestion mapping in a cloud. In: ACM PODC (2011)
8. Breitgand, D., Kutiel, G., Raz, D.: Cost-aware live migration of services in the cloud. In: SYSTOR'10 (2010)
9. Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., Shenker, S.: NOX: Towards an Operating System for Networks. In: ACM SIGCOMM CCR (July 2008)
10. Jung, G., , Hiltunen, M.A., Joshi, K.R., Schlichting, R.D., Pu, C.: Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures. In: IEEE ICDCS (2010)
11. Mann, V., Kumar, A., Dutta, P., Kalyanaraman, S.: VMFlow: Leveraging VM Mobility to Reduce Network Power Costs in Data Centers. In: IFIP Networking (2011)
12. McKeon, N., et al.: OpenFlow: Enabling Innovation in Campus Networks. In: ACM SIGCOMM CCR (April 2008)
13. Meng, X., Pappas, V., Zhang, L.: Improving the Scalability of Data Center Networks with Traffic-aware Virtual Machine Placement. In: IEEE INFOCOM (2010)
14. Sonnek, J., Greensky, J., Reutiman, R., Chandra, A.: Starling: Minimizing communication overhead in virtualized computing platforms using decentralized affinity-aware migration. In: IEEE ICPP (2010)
15. Stage, A., Setzer, T.: Network-aware migration control and scheduling of differentiated virtual machine workloads. In: ICSE CLOUD Workshop (2009)
16. Verma, A., Ahuja, P., Neogi, A.: pMapper: Power and migration cost aware application placement in virtualized systems. In: ACM/IFIP/USENIX Middleware (2008)