Consolidation of Underutilized Virtual Machines to Reduce Total Power Usage

SeyedAli Jokar Jandaghi
Dept. of Computer Science
University of Toronto
Toronto, Canada
saj@cs.toronto.edu

Arnamoy Bhattacharyya
ECE Department
Toronto, Canada
University of Toronto
arnamoyb@ece.utoronto.ca

Stelios Sotiriadis
ECE Department
Toronto, Canada
University of Toronto
s.sotiriadis@utoronto.ca

Cristiana Amza
ECE Department
Toronto, Canada
University of Toronto
amza@ece.utoronto.ca

ABSTRACT

Data centers (DC) have become one of the biggest markets with new challenges and opportunities in the past decade. Many big companies are owners of DCs providing services and cloud solutions. With this increasing demand and interest in cloud service, thousands of virtual machines (VM) are being instantiated to run a variety of services in a data center. Beside the benefit of service provisioning, a DC is a big consumer of electric power and producer of greenhouse gasses consequently. Because the VMs are not using their assigned resources all the time, resources are multiplexed via virtualization. However, current resource management methods are oblivious to actual utilization or power consumption. Monitoring of servers in big data centers like Google and Twitter has shown that the current resource utilization is less than fifty percent in total.

Specifically, in this paper, we use OpenStack, a popular cloud management software to orchestrate and manage VMs. In Open-Stack, the virtualization factor is a constant value oblivious to VM resource consumption. We design and implement two dynamic methods for adapting the virtualization factor based on the monitored VM resource consumption. In our first method, we identify VMs that are mostly idle, and we opportunistically migrate all idle VMs to one or more servers in such a way to keep the chance of performance degradation to a minimum, while saving total resources. In our second, more general method, we model consolidation of underutilized VMs as a knapsack problem. We show that our methods save a significant amount of underutilized resources while minimizing performance degradation during and after dynamic reconfiguration.

CASCON 2016 Oct 31 - Nov 2, 2016, Toronto, Canada

DOI: 10.1145/1235

CCS Concepts

•**Computer systems organization → Cloud computing;**

Keywords

Utilization; Cloud Computing; Data Centers

1. INTRODUCTION

Large-scale computing has been around for a while, and data centers (DC) are becoming a crucial part of the current and future digital technology era. A DC houses thousands of heterogeneous servers running Cloud solution services. Cloud computing is a compelling paradigm to deliver services over the internet. Cloud environments are highly dynamic environments, co-hosting several applications sharing resources in a multi-server environment. Uncontrolled resource sharing between co-hosted applications may result in violations of service-level requirements (SLA) for applications, thus creating a problem for service providers. Long-term resource over provisioning for the peak incidental load of all hosted applications is, however, unacceptable, due to the large cooling and power costs of large scale server platforms. Therefore, deriving approximate envelopes for application resource needs dynamically through VM monitoring and adjusting resource allocation on servers in DC dynamically is the only practical solution.

In this paper, we first present the common data center architecture, holistically as a system and then we investigate methods for more efficient resource utilization to reduce total power usage. Keeping in mind that one of the main electric power consumers of a typical data center are physical servers, our aim is to consolidate applications on fewer active servers and shut down idle servers in order to reduce total power consumption of the DC. Power consumption and total server infrastructure maintenance are the biggest portion of the total cost of ownership for a regular data center [1]. Out of this total power consumption, cooling of servers is the largest contributor and server electric consumption is the second largest source of energy usage [2]. Furthermore, monitoring of servers in big data centers like Google and Twitter has shown that the current resource utilization is

less than fifty percent total [3]. Moreover, an idle server may consume up to 60% of its regular power usage [4]. Finally, data centers are expected to use 139 billion kWh energy by 2020 [5], and the resulting CO2 emission are expected to be more than four times the current emissions by then [6]. In this paper we focus on methods for consolidating applications on fewer servers, hence allowing idle servers to be shutdown for conserving the most power.

Virtualization plays an important role in multiplexing the resources among multiple guest VMs in a Cloud platforms. In Cloud computing service environments, multiple users may share resources from a large pool of physical resources. Two examples of such physical resources are physical computing servers and the underlying network bandwidth. Users are unaware that they are sharing the resource with other users. They submit their jobs via the Cloud interface and can only measure the quality of experience for their submitted jobs. From their point of view, they own the entire resource, as the virtual-to-physical mapping is hidden from the user. The Cloud manager monitors each server status and provisions resources to submitted jobs based on its resource allocation algorithm. Some examples of these types of Cloud computing environments are Amazon EC2 and Microsoft Azure [7]. Therefore, among all the Cloud management techniques, resource allocation is becoming an increasingly important topic nowadays. Sharing physical resources across many users (with potentially different workloads) brings rise to several challenges. One such challenge is the inefficient resource utilization by each user while other users may be in need to those resources. Ensuring a correct resource allocation within a Cloud platform is vital to ensure the proper operations of each user's applications.

Some common mechanisms for controlling the resource allocation in Cloud platforms are: the flavor of the VM and virtualization factor (called Overcommit factor in OpenStack). The flavor of a VM can be selected from among a set of fixed predefined sizes such as large, tiny, or medium. VMs are allocated resources as they request them at creation time based on the flavor of VM but the actual physical resources allocated to a VM is controlled by the virtualization factor i.e the number of VMs that share the resource. Therefore, a VM asks for a list of resources according to its desired flavor and the Cloud manager determines the number of VMs sharing a resource.

In this paper, we introduce two methods for more power efficient resource allocation. The first method aims at consolidating the VMs that rarely use their resources which we call idle VMs. This method migrates multiple idle VMs into another server with very low chance of performance degradation supported by theoretical constructs. The second method is a more generic way to resolve the underutilization problem, by packing underutilized VMs together. The complexity of the first method is *O(nlogn)* for n virtual instances while the complexity of the second method is exponential for n virtual instances. Our first method is expected to run opportunistically for potentially a large number of idle VMs in underutilized data centers, with very low overhead.

Once the optimized mapping is established in either algorithm, we migrate the VMs to their destination server(s) which we call Recipient Server. The order in which VMs are selected for migration is specified by the proposed algorithms. However, we also consider the migration time as a secondary factor towards minimizing performance disruption.

In our evaluation, we used an installation of OpenStack and also real data from Google [8] for common resource usage of DC applications. For both methods, we measure the cost of migration in terms of time. We used real (Google) data for evaluating the second method. We evaluate both methods by the amount of saved resources.

The rest of the paper is organized as follow. We present the importance of energy efficiency in a data center in more details in section 2. Section 3 covers the design of our techniques and our main ideas to make a data center more power efficient through improving resource utilization. Technical details of our contributions are presented in sections 4 and 5. In section 6 presents our experimental evaluation. Finally, section 7 concludes the paper.

2. DATA CENTER AS AN ICT SYSTEM

Data centers are large scale connected computers together with other facilities providing the essential infrastructure necessary for providing information and communication technologies (ICT) services. This system is designed to manage heterogeneous hardware (servers) and allocated them to users. This service can be provisioning hardware with an operating system installed on that, namely platform as a service (PaaS), or it can be more abstract direct software services like monitoring (MaaS) or security (SaaS) âĂŐ[9]. Although Cloud paradigm services are different in their level of service but utilizing hardware to execute their jobs is common among all of them.

While a cluster is defined as a bunch of servers running and hosting many jobs under a single manager, it becomes more important to be utilized in a way to accommodate more jobs with adequate quality of service. The paradigm is similar to Google Borg architecture [10] and the cloud management software is assumed to be OpenStack. Jobs are defined as a group of tasks encapsulated in a single or multiple virtual instances. The virtual instance refers to virtual machines running an operating systems or containers running a lightweight operating system.

Virtualization is a foundation paradigm of cloud computing as it represents the benefit of sharing physical resources among multiple virtual instances (VMs). This sharing could be dynamic or static over time based on different management policies. The value of the virtualization factor should be inversely proportional to the level of activity of guest VMs. For example, two moderately active VMs each requesting n CPU cores can be assigned to a physical machine with only $n/2$ CPU cores and virtualization factor of 2.

Submitted virtual machine instantiation requests are getting processed by the cloud manager before being placed. After preprocessing, each request goes to the scheduling and placement which assigns physical resources to each virtual instance.

The scheduling and placement subsystem can work with different resource allocation granularity, coarse grain such as racks of servers or a network of servers and fine grain like memory cards and CPU chipsets.

An individual server, multiple servers within a rack, and servers under the same sub tree topology are other subsystems with different requirement and criteria. Elements of each level can show disparate features from other elements according to their technology, architecture, and spec-

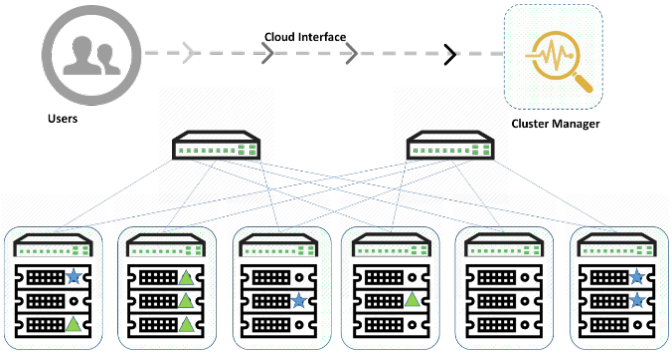

Figure 1: Leverage points to resolve underutilization issue.

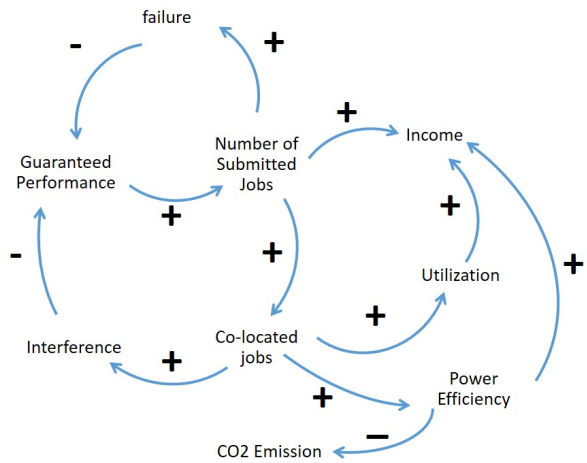

Figure 2: A causal loop diagram showing the essential links between server utilization, virtual machine QoS and income

ifications. 1 shows the different subsystems in a cluster of servers. Hard-ware/software heterogeneity of the servers is represented by different signs like stars, circles, and triangles. This heterogeneity can be due to different hardware technologies or required software packages on a server. Users can specify the hardware compatibility and desired type of server to host their virtual machines.

As for Control mechanism, a Gateway of data center uses the abstract features of each cluster to forward the submission request to the appropriate cluster. Each cluster has a central manager with backup manager(s) to be substituted in case of failures. The submitted jobs will be scheduled and placed by the scheduler based on the current status of the servers. The physical/virtual machine resource usage behavior is monitored by the manager and used to plan for next virtual instance placements.

Thus, monitoring each guest OS and analyze its behavior is quite essential. As a result, we can group them to idle and non-idle VMs with similar behaviors. Afterward, resource allocation should be revised. To do so, resource sharing factor between VMs should be assigned again. Reallocating VMs will be done using live migration which is handled by cloud management software like OpenStack.

Users specify the hardware dependencies and desired quantity of resources for their virtual machines unaware of which physical machine hosting their jobs. The data center gateway management distributes the submitted VM requests to its clusters. Each cluster manager checks the status of its associated machines and accepts to run the job if the requirements specified by user are satisfied in that cluster. Later it starts scheduling the virtual machine and assigns a server to host it. The manager keeps track of virtual instance resource usage to avoid underutilization whereas providing sufficient fair amount of resources to the VM in order to satisfy quality of service. In order to achieve this goal, it migrates, reschedule, or may kill some of the VMs if necessary.

This designed cluster of servers consists of multiple causal loops showing how each element within a cluster affects other elements. First, number of servers should be proportional to number of requests submitted to data center. Keeping in mind that VMs' lifetime and requirement is different, they have been categorized into two main groups. VMs running service jobs are interactive processes which are sensitive to delay and their lifetime is in order of hours to days whereas batch analytic jobs are much shorter and are not delicate to delay. It should be noted that service jobs are taking 60% of the cores in a Google Cloud platform [10]. Users usually over provision resources for their virtual instances and then submit their jobs through Cloud service provider interface to run their jobs. The job placement procedure is transparent to user, so user can only see the status of the job being changed from scheduling to running.

Secondly, scheduler time complexity algorithm should be proportional to the lifetime of the VM. Long time running VMs should be scheduled more conservatively since they are sensitive to delay. So the cost of scheduling should be aligned with VM's requirement and lifetime. Scheduling and placement will decide on co-locality of virtual instances which includes the trade off between performance and utilization. The more instances deployed on a server, the more chance of interference but at the same time more utilization. This interference and competition for resources can degrade the total performance and violate the quality of service for instances that it is hosting. Investigating different methods and policies used for scheduling instances is a crucial part of this analysis. Scheduling can happen once or be revised during the job's lifetime by migrating, killing, or cloning the virtual instance.

As any other business, the total cost of ownership for Cloud providers is a financial estimate determining their net income. 2 depicts the importance of utilizing infrastructure in an efficient way. Among the many factors that affect income, including application interference, failure, and utilization, in this paper we focus on resource utilization. We explore co-location of jobs as a way to optimize the total resource usage. Specifically, we propose methods to increase the total resource utilization by consolidating underutilized VMs onto fewer higher utilized servers while idle servers can be shutdown to conserve power.

3. PROBLEM DEFINITION AND MAIN IDEA

DCs are becoming a large factor in greenhouse gas emissions in the next decade due to their very high power consumption. An essential cause of this power usage is underutilized servers that host virtual instances. A server uses

more than half of its peak electricity consumption even if it does not run any jobs or host any idle virtual instance [4]. Hence, shutting down underutilized servers and utilizing fewer but more active functional servers is important. The underutilization is usually due to idle behavior of some of the virtual instances on servers.

Increasing utilization may come at the cost of performance degradation. Resource bottlenecks due to concurrent demands for a resource can cause performance degradation. Hence, violations of the guaranteed performance from the service level agreement (SLA) may occur. For example, Amazon Cloud solution, namely EC2, provides affordable virtual instances, namely Spot Instances [11], with a SLA. While the SLA may depend on the type of user and cost of the VM, in this paper we are assuming that any performance degradation induced by the power saving method should be negligible.

4. OVERVIEW OF FRAMEWORK

Keeping in mind that cloud has become very popular in academia and industry, many open source projects have been introduced to manage a cluster of servers by installing hypervisors on servers and managing connectivity of them. OpenStack is one of the platform management with large number of active contributors. It provides a comprehensive solution by installing hypervisors, network domains, and extra cloud features as separate projects.

OpenStack has a component responsible for provisioning resources to the instances, namely Nova. Nova uses the Nova scheduler to determine on which host a particular instance should be launched. Nova is also responsible for accommodation of newly launched VMs. The CPU âĂIJOvercommitâĂİ factor in a our OpenStack (Devstack) installation is set to 4 while the Memory sharing factor is set to 2. Nova scheduler would run the weight and filter algorithm [12] to find the suitable server for the desired flavor.

We implemented a component responsible for monitoring and collecting VM resource usage. It gathers the measured data to a database and provides access to the collected metrics. We provide a brief overview of each of these components to help understand our procedure

4.1 Nova

Nova is the fabric controller software of OpenStack to manage an IaaS. Nova has seven major components in which cloud controller is the center component connected to others. This component provides a global view of the cloud and is an interface between other components. Another crucial component is Nova-Scheduler which selects the suitable server to host a VM. Scheduler first apply Filter to determine which physical servers are eligible to be considered to host the VM. Then a Weighting process gives a weight to each of the hosts according to the costs relative to the request specifications. After the weighting step, a preferred sorted list shows the potential host. An abstract figure of Nova-Scheduler is shown in the figure 3. We will revise this procedure and reallocate the physical resources to degrade the total cost.

4.2 Monitoring and Data Streaming

We need an instrument to get integrated with OpenStack to collect measurements from cloud components. A python program does the monitoring and stream the results to our

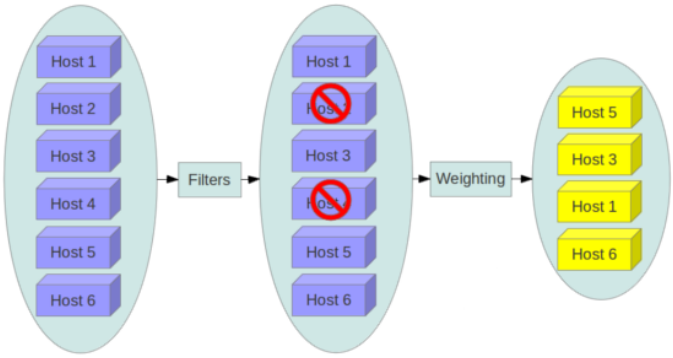

Figure 3: scheduling components in Nova [13]

database. It collects the desired measurements data by a component called Monitoring Collector and stores it in our database. Our script has another major component that provides an interface to be able to request recording and streaming the data. These commands can even be periodic to monitor for a specific alarm and record the result in separate files.

4.3 High-level Architecture

Figure 4 shows a high-level overview of our design to enable OpenStack and monitoring component to dynamically allocate resources to virtual machines. The following steps explain the procedure to classify and reallocate VMs in our platform:

1. Fetching all the VMs
   Nova provides API to query list of VM IDs as unique to each VM. We have used the command to fetch all the Unique Identification Number (UID) from the Nova's database.

2. Measurement
   In the OpenStack orchestration, our monitoring script is responsible for gathering measurement data and log them into it's data base. All the VMs has a monitoring and data streaming agent which monitor resource usage in individual VMs. We have used the monitoring tool to start recording each one of the VMs behavior and store their corresponding CPU Utilization value. Since the interval can vary based on configuration monitoring, we have used a fine-grained sampling rate to get all the details.

3. Algorithm
   Hitherto, we gathered all the required information about the VMs to classify them to idle and non-idle VMs. Idle VMs do not utilize their dedicated resources. The basic idea is to run the algorithm every one in a while during a day to alter idle tag of VMs. As a result, at the end of the day the algorithm would report which VM has been identified as an idle VMs. Now we have a list of VMs sorted by their average utilization. To consider the Service Level Agreement, we assign weights to each virtual machine based on the type of the user/application and then sort the final list.

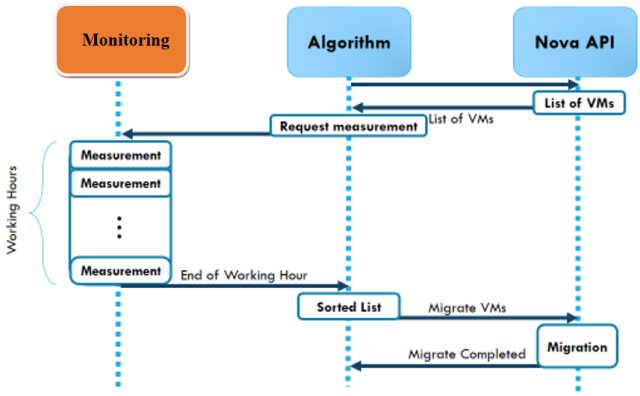

Figure 4: High level design

4. Migration and Consolidation
   The previous step provided a list of potential VMs to be migrated. Those VMs should be consolidated on another server with higher Overcommit factor. The total number of VMs that can be migrated is specified by the performance degradation bound on destination server which currently we set to one percent.

4.4 Low level Design

The low level schema shows more details about our approach to achieving efficient resource allocation within our platform. We have connected a server dedicated for migrating idle VMs, namely âĂIJRecipient serverâĂİ. Nova scheduler is been disabled on that server to avoid spawning any VMs on this server so that it becomes dedicated for their level of activity. After collecting the measurements during one working day, we sort the VMs according to the idle VMs. It is more likely to migrate idle VMs instead of utilized VMs. We have also implemented a prioritization scheme for migrating VMs belonging to different users. For instance, idle VMs belonging to high priority users will not be migrated whereas lower priority users VMs will be migrated. We scale the sorted list of VM CPU Utilization from the idlest to the least idle by a weighting factor to ensure that high priority users VMs are least likely to be migrated compared to lower priority users.

Algorithm 1: Idle VM Consolidation Pseudo-code

Input: Type and number of VMs
Output: An Array of Migrated VMs
Create_Virtual_Machines(number, flavor);
Run_Monitoring (interval, Duration);
Sorted_Array_of_vms= Sort(SLA, Average Utilization);
Migration (Sorted_Array_of_vms);

The Pseudo code of our technique is represented in algorithm 1. We have created an experiment by spawning VMs of two different users in the Create_Virtual_Machines script. Then we gather the monitored values of those VMs periodically and do the sorting base on the owner of the VM at the Sort script. At the end we used the Nova API to do the migration to destination agent up to a limited number of VM.

4.5 Performance Modeling

In this Section, we want to show how performance of the idle VMs might be affected by migration. It is obvious that sharing resources would decrease the performance by some factors but we would see if this factor is negligible or not. We formulate our problem as follow:

We are planning to migrate n VMs to a desired agent. Each VM has registered for R resources and for the sake of simplicity we assume all the VMs require same amount of resources. We define a random variable W_i for the VM i showing its usage of the dedicated R resources. Resource usage can get values from zero to R at each time slot. We would represent the usage in terms of percentage by W_i.

W_i is the random variable showing the utilization percentage of the ith VM. W_i can get values between 0 and 1 while in our case the W_i's of migrated VMs is between zero and threshold. Idle VMs or W_i's are independent since each VM is used by different users and we assume they follow a roughly similar distribution.

In the destination host we have C resources that will be shared among VMs that each needs R resources. The sharing factor is shown by $\frac{R*numberofVMs}{C}$.

Total demand at time t is the summation of all the VMs' corresponding W_i(t) to be shown by S_n(t). We know that each of the VM behavior is independent and since they have been filtered to fulfill the threshold requirement, they follow the same distribution. Our goal is to find out the probability that the total demand S_n exceeds the total number of resources C. Although according to the Central Limit Theorem [14] S_n can be estimated with a Normal distribution but since we want to compare the performance degradation for rare events, we should use another approach.

We are interested to find out the rate of decrease in performance by increasing n. In order to do so, we use the Chernoff bound [14] which specifies the boundary for random variable Y as follow:

$$\Pr(Y \geq 0) \leq \mathrm{E}\left[e^{\theta \cdot Y}\right]. \tag{1}$$

$$\Pr(\sum_{n=1}^{n} w_{i(t)}) - C \geq 0) \leq \mathrm{E}\left[e^{\theta \cdot (\sum_{n=1}^{n} w_{i(t)}) - C)}\right]. \tag{2}$$

$$\mathrm{E}\left[e^{\theta \cdot (\sum_{n=1}^{n} w_{i(t)}) - C)}\right] = e^{\theta \cdot C}\mathrm{E}\left[e^{\theta \cdot (\sum_{n=1}^{n} w_{i(t)}))}\right]. \tag{3}$$

Also, we know that the Moment Generating Function (MGF) of a random variable Y is defined by this formula.

$$M_Y(\theta) = \mathrm{E}\left[e^{\theta \cdot Y}\right]. \tag{4}$$

Logarithm value of the moment function is called Cumulant generating function expressed by $\Lambda_Y(\theta)$

We rewrite the Chernoff bound with respect to MGF of W_i as follow:

$$\Pr(\sum_{n=1}^{n} w_{i(t)}) - C \geq 0) \leq e^{\theta \cdot C - n \cdot \ln M(\theta)}. \tag{5}$$

$$\tag{6}$$

We name the argument $\theta \cdot C - n \cdot \ln M(\theta)$, Rate function and

to find the lower bound for the above inequality, we should find the maximum value of Rate function as a function of θ. We assume that the acceptable chance of performance degradation is equal to 1 percent which means that the rate function is equal to R. The rate function has two unknown variables: θ and n. $\theta*$ is the value that maximizes the rate function. We specify the W_i as follow.

$$w_i = \begin{cases} 1, & \text{with probability } \frac{1}{m} \\ 0, & \text{with probability } \frac{m-1}{m} \end{cases} \tag{7}$$

As for the moment generating function we have:

$$M_{w_i}(\theta) = \mathrm{E}\left[e^{\theta \cdot Y}\right] = \frac{m-1}{m} + \frac{1}{m} \cdot e^{\theta} \tag{8}$$

Hitherto, we can rewrite the function and find the $\theta*$ expressed as follows:

$$\begin{aligned} Rate - function &= \theta \cdot C - n \cdot \ln M(\theta) \\ &= \theta \cdot C - n \cdot \ln(\frac{m-1}{m} + \frac{1}{m} \cdot e^{\theta}) = R \end{aligned} \tag{9}$$

The above equation does not have any closed form answer. We can write the rate function as the function of n and find the n based on θ *in our scenario and find corresponding answers.

5. CONSOLIDATION TO RECIPIENT SERVER

5.1 Performance Modeling for idle VM Consolidation

Virtual Machines are been assigned dedicated virtual resources from hypervisor. This allocation is based on the sharing factor of hypervisor and the virtual machine placement algorithm. For OpenStack, by default, this allocation is constant during a VM life time. To illustrate, if a VM asks for x number of resource r, the server will be assigning x Overcommit factor to VM. The sharing factor is going to be constant for many legacy methods. This virtualization is hidden from user and user can assume running the virtual machine according to requested resources. If the virtualization sharing factor is equal to one, it means the server resource is exclusively reserved for the VM, otherwise, the resource is shared among co-located VMs. In a real scenario, many VMs are underutilizing their resources due to overestimating of required resources from user. This underutilization could be true for any type of resources like memory, CPU, and IO. We propose migration of those VMs to another server, namely *Recipient server*, with higher sharing factor to release more physical resources on source server. This resource saving comes at the cost of migration and higher chance of interference for VMs in destination server. The interference is due to higher density of VMs in the Recipient server.

5.2 General Modeling for underutilized VMs:

Knapsack problem (KP) is a common usage problem in combinatorial optimization that describes a strategy to select a number of items according to their values and size to put them in a knapsack with limited size. The constraint is not exceeding the total size of the knapsack and the objective

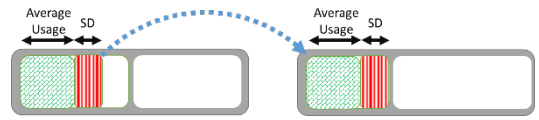

Figure 5: Releasing unused resource for a VM according to its statistical usage

is to collect a set of items with highest possible value. Following, we map all the elements of our consolidation problem to model it as a multi-dimensional knapsack problem.

We assume each VM will be having the same size as average plus standard deviation for either one the resources. As for value of the VM, we consider type and migration cost of the VM. Some VMs are having specific priorities due to their users or running software specified by their type. For example, a VM could be running a critical process that needs not to be disrupted. In addition to VM type, VMs have different migration costs due to their size, memory usage, and network link capacity. We formulate the value of each VM to be used in our knapsack problem algorithm.

Size of VMs: As for the sharing factor of VMs in Recipient server, we suggest a method to reconsider size of the VMs from Recipient server's hypervisor perspective without revealing any change to user. This reallocation of resources is based on the behavior of the VM in the last day. We basically replace the size of a VM that asked for x number of resources with average of observed values plus the observed standard deviation. The mean is the least expected amount of resources that we have observed the VM is using it. The Standard deviation is an added margin to decrease risk of resource outage for the VM.

As an example, a VM asks for 4 cores of CPU, 8GB of memory in addition to a 100Mb of network link and we figure out the resource usage of VM to be less than assigned resources. For the sake of this example, say the VM is using half of the resources in average with bouncing a quarter off the expected usage. It means the VM was mostly idle for the fourth quarter of its resources. As shown in figure 5 , more resources will be available for potential VMs by resizing the VM based on its past behavior.

5.3 Multidimensional Knapsack Problem

Knapsack problem is one of the famous challenges in combinatorial optimization. The name of knapsack is obtained by the following well known example. Suppose a hitchhiker needs to fill a knapsack for a trip. There are n items available to select and each of them has a pro t value of p_i, i.e., the degree of the usefulness of this item during the trip, as well as a size of w_i. A natural constraint arises that the aggregated size of all selected items cannot exceed the capacity of the knapsack, denoted by c. The objective of the hitch-hiker is to select a subset of items while maximizing the overall value under the capacity constraint. To model this decision process, we introduce a binary decision variable x_i for each item where $x_i = 1$ if the ith item is selected and packed while $x_i = 0$ otherwise. The standard knapsack problem is given by:

$$max \sum_{i=1}^{n} p_i x_i \tag{10}$$

such that

$$\sum_{i=1}^{n} w_i x_i \leqslant c, x_i \in \{0,1\} \forall i \tag{11}$$

The multidimensional knapsack problem is an extension of the standard knapsack problem. It is given by the following equations:

$$max \sum_{i=1}^{n} p_i x_i \tag{12}$$

such that

$$\sum_{i=1}^{n} w_{i,j} x_i \leqslant c_j, x_i \in \{0,1\} \forall i, \forall j \in M \tag{13}$$

n items have to be placed in the knapsack, according to its capacities $(c_1, \ldots, c_m)$, $m \in M$. To an item $j \in N = \{1, 2, \ldots, n\}$, the following variables and vectors are associated:

- The decision variable $x_i \in \{0,1\}$, $x_i = 1$ if the item i is placed in the knapsack, and $x_i = 0$ otherwise.
- The profit $p_i \geqslant 0$ and
- The weights $w_{i,j} \geqslant 0$, $j \in M = \{1, \ldots, m\}$.

5.4 Performance Modeling

Values of VMs: Since we can only change the number of virtual resources on a separate server, we need migration of VMs. But this migration comes at some cost due to overhead on the platform and disruption to the user. Therefore, we need to specify the cost of the migration for each VM based on the type and location of the VM. In another word, we can define value of each VM to be in reverse order with its migration cost. Migration feature is used as a means to allow reshuffling of VM's for proper resource management. We briefly introduce migration types first and then go over the cost model for VM migration to determine value of any VM.

5.5 VM Migration

VM migration is mainly a transfer of the memory content and CPU registers (and maybe disk) of the VM which contains information of the processes running on the VM from one host to another. There are different methods to migrate VMs regarding whether the storage among servers is shared or not, and if the VM is allowed to be disrupted during the migration phase. Two common methods used for migration are: Stop-and-Copy and live-migration. Both of the methods needs to copy the disk in case no share network storage is available but the discrepancy comes at copying the content of memory.

Stop-and-Copy stops the process inside the VM and copies the memory to destination server. This stop cause a disruption to VM process during the migration time. After the migration, they process may need time to recover its state. Whereas the live-migration consists of two phases: pre-copy phase where the transfer is carried out in cycles, and in each cycle, the pages that got dirtied since the last cycle are copied again. In the pre-copy phase, the applications running are live. The next phase is downtime-phase where at some point the VM is paused and the last batch of dirty pages is transferred. After this phase the VM's can start over at the target host. The benefit of live-migration over stop-copy migration is reduction in downtime, but doing so comes from trade off of total memory volume transferred which is clearly high in live-migration. The time taken to transfer this higher memory volume also in-creases correspondingly.

5.6 VM Migration Cost

The applications running in the VM's would be better off if there was no disruption of service, and so would the cloud solution, as business would be running smoothly. As discussed earlier, VM migration cost is in reverse proportional order with VM value in out modeling. So it's necessary to discuss the factors that affect the migration process and model VM migration cost according to disruption time.

Copying the disk is possible in case of no shared storage is available but memory migration is different for different migration methods. Akoush et al [15] showed that the page dirty rate and link speed are the major factors influencing migration times. Total traffic generated for VM migration depends on the page dirty rate and the total migration time depends on both capacity of link and the total generated traffic. While the total traffic generated for stop-and-copy is simply the same as size of memory, generated traffic for live-migration is equal to memory size plus data migrated in each pre-copy phase. [16] provided a proof that the total generated traffic is as:

$$N = D + N_0 + \sum_{i=1}^{n+1} N_i = D + M + M \cdot \frac{1 - (\frac{R}{L})^{n+1}}{1 - \frac{R}{L}} \tag{14}$$

Where D is disk size, N_0 is the entire memory size, and each N_i is traffic generated on pre-copy cycle i. R is the VM's page dirtying rate, L is the link capacity, and n is the number of pre-cycles calculated as below:

$$n = \min(\lceil \log_{R/L} \frac{T \cdot M}{L} \rceil, \lceil \log_{R/L} \frac{X \cdot R}{M(L-R)} \rceil) \tag{15}$$

Where T is defined to be a user setting in VMware vMotion technique, denoting the switchover goal time. X is showing the threshold for difference before and after a pre-copy cycle. It is set by user to be called the minimum required progress amount. Now we can say total time required for migrating a VM is equal to N/L. Thus, the higher the time for migration, the lower the VM value is.

In a nutshell, we formulate the problem of underutilized VM consolidation to knapsack problem by expounding VM size and value with constraint of the Recipient server resources. There are number of VM's, $(V_1, V_2, , V_h)$ running on multiple hosts. Each VM k has been assigned capacity C_k, including all resources, initially on source server although it may not use all the provisioned resources. We interpret the VM's actual size to be S_k equal to average resource usage in addition to standard deviation. As for VM value V_k, we consider the VMs with less time migration overhead to be more valuable. This time is calculated on previous part. The pseudo code for this modeling is provided in algorithm 2.

6. EVALUATION

Algorithm 2: Underutilized VM consolidation Pseudo code

```
Input: collected monitoring data of all VMs
Output: A list of VMs to be migrated to Recipient
        Server
for k in list of All VMs do
    for R in list of all resources do
        Avg_k,R =Calculate_AVG_VM_usage(R, K)
        Standard deviation_k,R =
        Calculate_SD_usage(R,K) Size_k,R=AVG_k,R+
        Standard deviation_k,R Size_k= Size_k+size_K,R
    end
    MT=Calculate_Migration_Traffic(T,X,L,R,M,D)
    Time_k=MT/L Value_k=1/Time_k
end
Knapsack(Value, Size, Recipient Host capacity)
```

Table 1: Migration time for different VM flavors

VM flavor	OS Name	Virtual Core	Memory	Disk (GB)	Migration Time(seconds)
Tiny	Cirros	1	512MB	1	22
Small	Cirros	1	2GB	10G	23
Medium	Cirros	2	4GB	10	55
Large	Cirros	4	8GB	10G	79

We setup an OpenStack platform for our experiments. It consists of two 8 Cores Intel Xeon servers with 32 GB of main memory on each to run as controller and compute node on OpenStack. The servers have QEMU hypervisor installed. We enable live migration on servers to test the performance of migration on our platform. We run couple of VMs and then run YCSB Cassandra workloads on them besides leaving couple of them unused. We collect the monitored data of their resource usage for all the VMs. In our problem, the destination server can accommodate 16 VMs that request 2 virtual CPUs with sharing factor one. It means that the total number of CPU resources is equal to 32 in our experiments.

We test the migration speed for single and concurrent VMs to estimate the time of migration. This test is done for VMs running Centos with flavor tiny. We wrote a python script to capture the status of VMs from nova and measure the time in between the VM state changes from migration to active.

As we can see from the results shown in Table 3, the overhead for migration of idle VMs is negligible as compared to the amount of resources that can be saved. As an example, the default configuration of our OpenStack platform supports up to 32 of those tiny VMs as each VM would be assigned 1 virtualized core of CPU. This number can be increased based on the observed distribution of VMs. We observe a Bernoulli distribution with probability of success equal to 1/6 (or m=6 in the probability equation 7).

Assuming the VMs behave like before, and other resources are sufficient to host them. Based on Chernoff bound, equation 6, the number of VMs with the aforementioned observed behavior that can be consolidated is 83 meaning we can have 83 VMs with less than 1% risk of performance degradation. This result is calculated from the Chernoff bound inequality and shows savings of more than half of all resources.

Consolidation of Underutilized VMs to Recipient Server: We used Google data recorded for a day in a cluster to build our model. Google data was released in 2011 by Google [8] with detail information for the resource usage of millions of Google containers in one Google cluster. Instead of containers used by Google we use virtual machines (VMs) to get generality in our solution. A Google cluster is defined to be a set of networked racks of servers. Google runs each separate set of processes in a container called tasks. We used resource requests and usages for both memory and CPU. The resource usage samples have time stamps of every five minutes and we collect 280 time stamps for 383 Google containers.

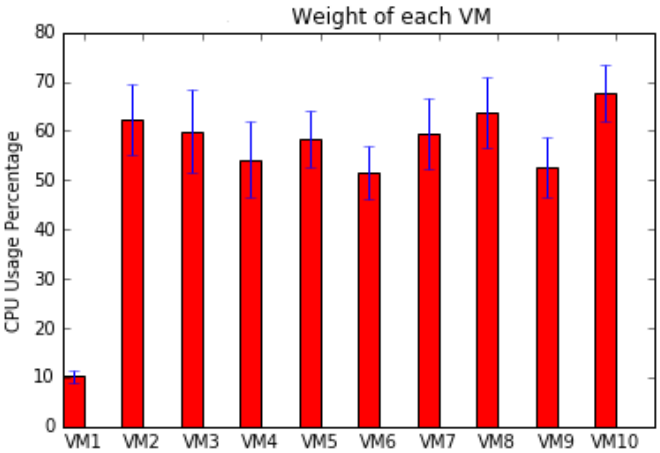

Figure 6: Average and standard deviation CPU usage for 10 random VMs

To get other parameters involved in our model we considered two resources: CPU and Memory. We synthetically assign them to different servers with different link capacity to the Recipient Servers.

Standard deviation and Average usage of both CPU and memory are computed for all the VMs. These observations for a few VMs are shown in figure 6,7, and 8. These figures show monitored utilization of resources versus all requested resources associated with all virtual instances in our selected portion of Google data. This monitored utilized value is the size of the VMs for memory and CPU representing the average resource usage behavior of the VMs plus the standard deviation of the resource usage.

Some resources will be saved if underutilized VMs migrate to the Recipient server which has higher Overcommit factor. This means the density of those VMs in Recipient server is higher than source server(s). We defined a metric called $saved-resource$ to measure the improvement. It measures the difference between value of initial assigned resources to VMs and the revised assign resources to them. This improvement is highly dependent on the configuration of the Recipient server and the values of VMs. To show this connection, we run the experiment for different configurations of Recipient server in both with/without VM values.

As shown in figure 9, best savings come at the cost of having a bigger Recipient server. Usually, the potential underutilized candidates are much more than the capacity of potential server. Our result has shown that for a recipient server with 11 cores of CPU and 7 GB of memory, the to-

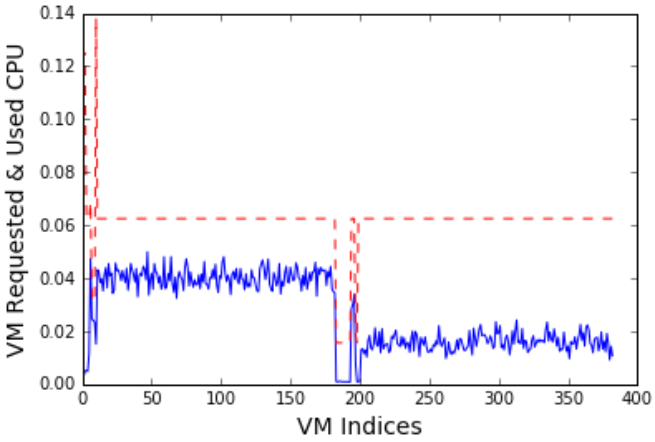

Figure 7: Requested CPU (dotted line) and utilized CPU for all VMs

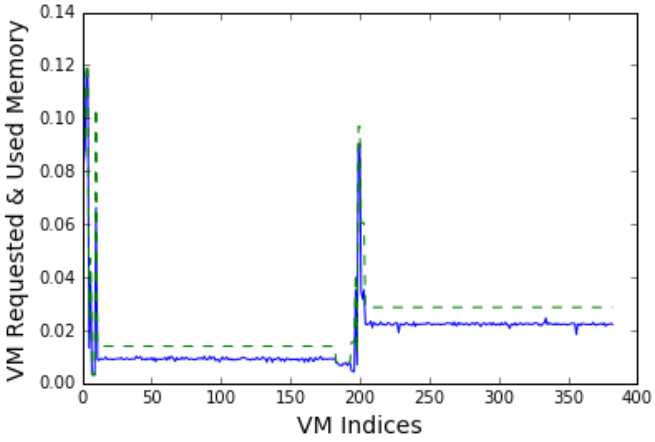

Figure 8: Requested memory (dotted line) and utilized memory for all VMs

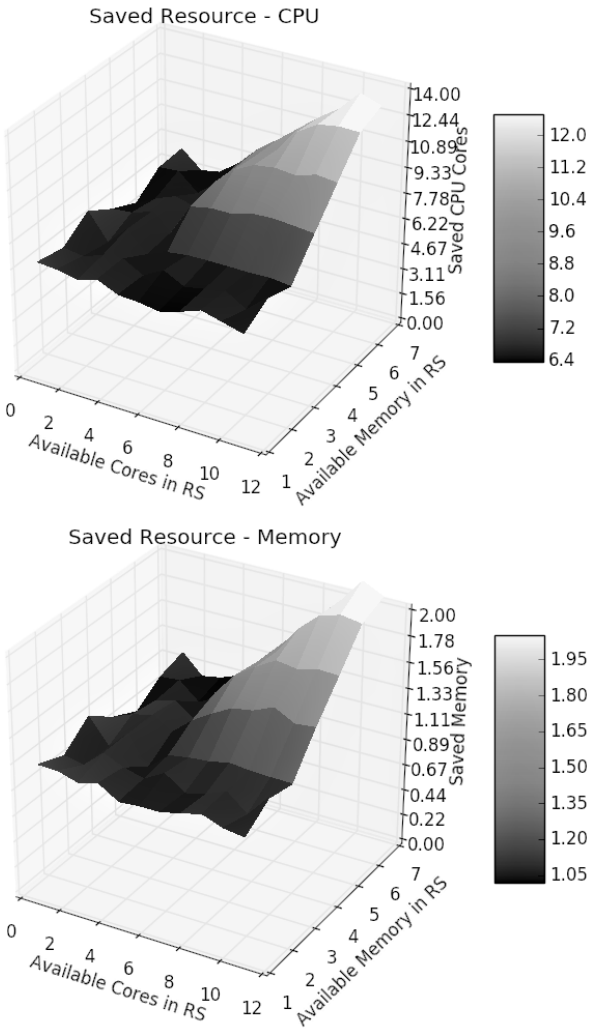

Figure 9: Saved resources for CPU and memory in case of VMs with different priorities

tal saved resource for CPU and memory is roughly 13 and 2. We observe that the total reserved CPU is almost 23 cores and for memory is 8GB. This shows an improvement of more than 50% in CPU saving and 25% in memory saving. The same diagrams are presented in figure 10 for the case of knapsack modeling with similar values.

7. CONCLUSION

In most current Cloud platforms, users ask for a certain VM specifications to run their jobs. The Cloud manager dedicates virtual resources for each job and places the job on a certain host based on its resource provisioning algorithm. Sharing hardware resources across multiple jobs can improve resource utilization but, at the same time has the risk of application interference, hence performance degradation. A common solution is to overprovision the re-sources which may waste electricity for running and cooling too many physical servers.

In this paper we investigate methods to increase resource utilization with the least risk of performance degradation. We introduce and evaluate two methods to tackle the resource underutilization problem. The first method is a simple method that sorts VMs based on their average CPU utilization and then models the idle VMs to be migrated with negligible risk to performance. The second method is a more generic method for packing underutilized VMs that models the VM placement problem as a multidimensional knapsack problem. The complexity of the first method is O(nlogn) for n virtual instances while the complexity of the second method is exponential for n virtual instances (since the solution to the knapsack problem is known to be NP complete).

Therefore, we expect that our first method will be run opportunistically in underutilized data centers with very low overhead. Our results for both methods, based on real Google data show that a significant number of resources (up to half of the total resources) could be saved by consolidating underutilized VMs onto fewer servers.

8. ACKNOWLEDGMENTS

Thanks to Dr. Hadi Banazadeh for his help on running

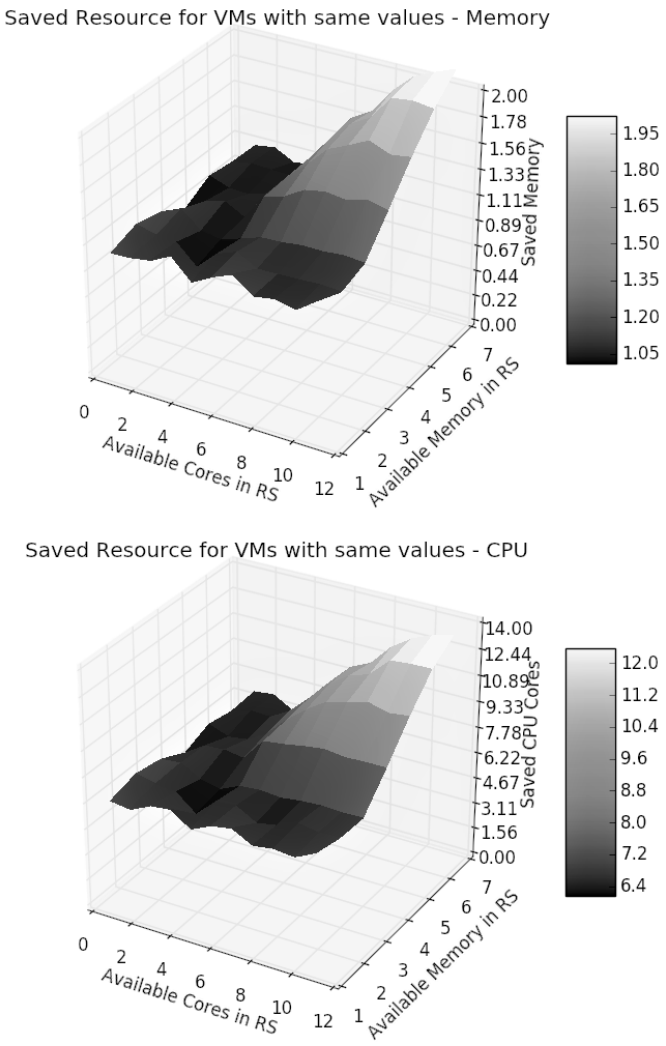

Figure 10: Saved resources for CPU and memory in case of VMs with similar priorities

OpenStack experiments.

9. ADDITIONAL AUTHORS

10. REFERENCES

[1] K. Vaid, "Datacenter power efficiency: Separating fact from fiction," in *Invited talk at the 2010 Workshop on Power Aware Computing and Systems*, vol. 1, 2010.

[2] B. Weihl, E. Teetzel, J. Clidaras, C. Malone, J. Kava, and M. Ryan, "Sustainable data centers," *XRDS: Crossroads, The ACM Magazine for Students*, vol. 17, no. 4, pp. 8–12, 2011.

[3] L. A. Barroso, J. Clidaras, and U. Hölzle, "The datacenter as a computer: An introduction to the design of warehouse-scale machines," *Synthesis lectures on computer architecture*, vol. 8, no. 3, pp. 1–154, 2013.

[4] C. Delimitrou, "Improving resource efficiency in cloud computing," Ph.D. dissertation, Citeseer, 2015.

[5] P. Thibodeau. (2014) Data centers are the new polluters. [Online]. Available: http://www.computerworld.com/article/2598562/data-center/data-centers-are-the-new-polluters.html

[6] S. Bouton, J. Creyts, T. Kiely, J. Livingston, and T. Nauclér, "Energy efficiency: A compelling global resource," *McKinsey Sustainability & Resource Productivity*, 2010.

[7] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of internet services and applications*, vol. 1, no. 1, pp. 7–18, 2010.

[8] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: format+ schema," *Google Inc., White Paper*, pp. 1–14, 2011.

[9] A. Al-lawati and A. H. Al-Badi, "The impact of cloud computing it departments: A case study of oman's financial institutions," in *2016 3rd MEC International Conference on Big Data and Smart City (ICBDSC)*. IEEE, 2016, pp. 1–10.

[10] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at google with borg," in *Proceedings of the Tenth European Conference on Computer Systems*. ACM, 2015, p. 18.

[11] O. Agmon Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir, "Deconstructing amazon ec2 spot instance pricing," *ACM Transactions on Economics and Computation*, vol. 1, no. 3, p. 16, 2013.

[12] R. Nasim and A. J. Kassler, "Deploying openstack: Virtual infrastructure or dedicated hardware," in *Computer Software and Applications Conference Workshops (COMPSACW), 2014 IEEE 38th International*. IEEE, 2014, pp. 84–89.

[13] "Nova scheduler," http://docs.openstack.org/kilo/config-reference/content/section_compute-scheduler.html, accessed: 2016-05-30.

[14] S. Keshav, *Mathematical foundations of computer networking*. Addison-Wesley, 2012.

[15] S. Akoush, R. Sohan, A. Rice, A. W. Moore, and A. Hopper, "Predicting the performance of virtual machine migration," in *2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. IEEE, 2010, pp. 37–46.

[16] 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 *International Conference on Research in Networking*. Springer, 2012, pp. 190–204.