Towards an understanding of oversubscription in cloud

Salman A. Baset, Long Wang, Chunqiang Tang
IBM Research

ABSTRACT

Oversubscription can leverage under utilized capacity in the cloud but can lead to overload. A cloud provider must manage overload due to oversubscription for maximizing its profit while minimizing any service level agreement (SLA) violations. This paper develops an understanding of oversubscription in cloud through modeling and simulations, and explores the relationship between overload mitigation schemes and SLAs.

1. INTRODUCTION

Cloud providers oversubscribe their data centers to leverage unused capacity and to maximize their profits. When a cloud is oversubscribed, overload can happen. For a cloud provider, the key to maximizing profits is to effectively mitigate overload while meeting SLAs promised to the customers. If an SLA is violated, a provider may have to reimburse the customer for any violations. The mechanisms for mitigating overload and the SLAs promised to the customer determine the extent to which a provider can oversubscribe its cloud for maximizing its profit, and how it should configure and manage spare capacity for mitigating overload. The goal of this paper is to develop an understanding of oversubscription in cloud and to explore the relationship between oversubscription objectives (i.e., profits), overload mitigation schemes, and SLAs, through modeling and simulations.

The rest of the paper is organized as follows. Section 2 describes the problem setting. Section 3 gives an overview of oversubscription including overload mitigation mechanisms and service level agreements (SLAs). Section 4 provides a theoretical basis for oversubscription problem and notes that it is a variant of online multiple constraint multiple knapsack problem. Section 5 presents simulations and results for evaluating various aspects of oversubscription problem, such as the relationship between overload detection interval and resource request interarrival time, and the performance of VM quiescing and live migration schemes for mitigating overload vs. a do-nothing approach. Section 6 lists some of the questions that we are investigating as part of ongoing work.

2. PROBLEM SETTING

We consider an Infrastructure as a Service (IaaS) cloud provider which lets customers run multiple virtual machines hosted on its physical machines (PMs) (or physical hosts). The IaaS provider is multitenant, i.e., it runs workloads from different customers. Although, we focus on IaaS providers in this paper, we believe that our analysis can be extended to other cloud service models such as Platform as a Service (PaaS) and Software as a Service (SaaS). The exploration is part of ongoing work.

A cloud provider can configure its physical and virtualized infrastructure in several ways. One key aspect of this infrastructure is whether the virtual machines and their virtual disks are located on the same physical host or whether the virtual disks are hosted on a shared SAN storage. If the virtual disks are hosted on a local storage, then the complete virtual disk does not need to be migrated to another physical host during overload, which is a costly network operation, and makes live migration time consuming and prohibitively expensive. On the other hand, if the virtual disks are hosted on shared storage, only the live memory footprint of a virtual mahine needs to be migrated to another physical host. For this paper, we assume that virtual disks are hosted on a shared SAN storage.

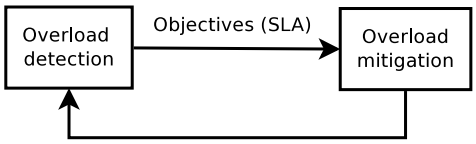

Figure 1: Conceptual overview of handling overload in an oversubscribed environment.

An IaaS cloud provider lets its customers buy virtual machines of different 'size'. Amazon EC2 [1] lets user purchase small, medium, or large VMs (or instances). In this paper, we consider that all VMs running on a PM are of the same size. As part of ongoing work, we are exploring the relationship between running VMs of different sizes on an oversubscribed physical machine.

Assume a central scheduler similar to VMware's DRS [7] is configured with the overload mitigation policies and continuously monitors the performance of VMs. If a VM or PM starts suffering from overload due to oversubscription, the scheduler kicks in overload mitigation that are guided by VM SLAs as discussed in Section 3.3.

3. OVERSUBSCRIPTION OVERVIEW

Overload can happen in an oversubscribed cloud. Conceptually, there are two steps for handling overload, namely, detection and mitigation, as shown in Figure 1.

A PM has CPU, memory, disk, and network resources. Overload on an oversubscribed host can manifest for each of these resources. When there is memory overload, the hypervisor swaps pages from its physical memory to disk to make room for new memory allocations requested by VMs. The swapping process increases disk read and write traffic and latency, causing the programs to thrash. Similarly, when there is CPU overload, VMs and the monitoring agents running with VMs may not get a chance to run, thereby increasing the number of processes waiting in the VM's CPU run queue. Consequently, any monitoring agents running inside the VM also may not get a chance to run, rendering inaccurate the cloud provider's view of VMs. Disk overload in shared SAN storage environment can increase the network traffic, where as in local storage it can degrade the performance of applica-

tions running in VMs. Lastly, network overload may result in an under utilization of CPU, disk, and memory resources, rendering ineffective any gains from oversubscription.

Overload can be detected by applications running on top of VMs, or by the physical host running the VMs. Each approach has its pros and cons. The applications know their performance best, so when they cannot obtain the provisioned resources of a VM, it is an indication of overload. The applications running on VMs can then funnel this information to the management infrastructure of cloud. However, this approach requires modification of applications. In the overload detection within physical host, the host can infer overload by monitoring CPU, disk, memory, and network utilizations of each VM process, and by monitoring the usage of each of its resources. The benefit of this approach is that no modification to the applications running on VMs is required. However, overload detection may not be fully accurate.

Another aspect of overload detection is the transient and permanent nature of overload. Williams *et al.* [9] observe memory overload to be transient for SPECweb like workloads. In Section 5, we explore the relationship between overload detection interval, and interarrival rate of resource requests that can cause overload.

3.1 Mechanisms for Mitigating Overload

There are several mechanisms for mitigating overload.

- **Stealing** allows an hypervisor to 'steal' resources from other underloaded VMs running on the same PM, and give them to an overloaded VM(s). As an example of stealing, VMWare ESX server [8] was the first to describe memory balloon drivers which allow removing memory pages from one VM and give them to a needy VM. When an overload is detected, this is the first strategy a hypervisor can employ for mitigating overload.

- **Quiescing VMs** can be used to shutdown VMs on an overloaded PM. The shutdown VMs can then be restarted when PM no longer suffers from overload, or can be migrated offline to an underloaded PM.

- **Live Migration** allows a provider to migrate a set of VMs from an overloaded PM to an underloaded PM. Live migration is a good solution when the storage of VMs is located on a SAN-like device. This is because only the memory footprint of VM needs to be migrated. When VM storage is local, live migration may be prohibitive due to heavy utilization of data center network for migrating the VM disk. A mechanism like streaming disks can be used in such scenarios (see below).

- **Streaming Disks** is similar to live migration except that the complete transfer of disk is not necessary. Instead, only a 'small' portion of VM's local disk is transferred to an underloaded PM which is sufficient for the VM to start. Once the VM is started on the underloaded PM, it can continue to access its remaining disk on the original PM. The remaining disk is then transferred when the network load connecting the PMs involved is low. FVD [4] is an example of a streaming disk solution.

  If a streaming disk solution is not managed carefully, the data center disk space can quickly become fragmented. That is, a number of migrated VMs may be accessing their disks over network, crisscrossing the data center network, and causing a network overload. However, streaming disks is an attractive solution when virtual disks and VMs are on the same PM.

- **Network Memory** allows a provider to utilize memory of another machine as a swap space over the network. This mechanism has the advantage that it can potentially alleviate load on the local disk of the physical machine due to swapping. Note that network memory is still more expensive than physical memory access (nano second vs. micro second), but swapping to disk is more expensive than network memory (micro second vs. millisecond for spinning disks).

  Similar to streaming disks, network memory also suffers from data center fragmentation.

  In this paper, we assume that stealing has failed to mitigate overload, and therefore additional mechanisms are required. Of the schemes discussed above, we only consider 'quiescing VMs' and 'live migration' for overload mitigation in this paper.

3.2 Constraints for Overload Mechanisms

A user or provider of VMs may specify constraints when quiescing or live migrating a VM. The constraints will likely reflect the interdependency among different components of an application running in different VMs. As an example of a user specified constraint, a VM may only be migrated if all other VMs that are running other components of the application are also migrated. The provider may specify a constraint for administrative or management reasons, i.e., a VM may only be migrated or use network memory from a PM in the same physical rack. The constrains can be specified using a placement restriction matrix such as the one used in [5].

3.3 Service Level Agreements (SLAs)

Service level agreement codifies the level of service that a cloud (or any) provider promises its customers. It is the sole compensation remedy for a customer if it does not receive the promised service. SLAs are equally important for providers and consumers for cloud services. For cloud providers, SLAs promised to the customers can determine the extent to which a provider can oversubscribe its cloud while meeting its SLAs. The definition of overload and its detection is intrinsically tied to the SLA promised to the customer. For example, an SLA may state that a VM is considered unavailable if it is unable to request the provisioned resources for a duration of five minutes, and SLA is violated if such violations occur more than 0.1% of time. Thus, a provider must perform overload detection and mitigation so that aggregate violations never exceed 0.1% of the time.

What type of SLAs do the public cloud providers promise their customers? A cursory look at the SLAs of Amazon EC2 [2], Rackspace Cloud Servers [3], and Microsoft Azure [10] reveals that these providers do not offer any performance guarantees on the VMs and instead only provide uptime guarantees. The uptime guarantees are also weak. In case of Amazon EC2, the uptime guarantee is on a per data center

basis, i.e., a data center is considered unavailable if a customer cannot access any of its instances in an Amazon data center or launch replacement instances within the same data center for a contiguous interval of five minutes. Rackspace implicitly offers uptime guarantee on a per VM basis, but similar to EC2, it does not offer any performance guarantees. However, even if these providers do not promise any performance SLAs, they must internally manage resources per VMs so that the uptime SLA is less likely to be violated.

3.4 Group SLAs

Performance vs. uptime is one dimension along which the SLAs can be considered. Another aspect is whether the performance or uptime SLAs are promised on a per resource or a group of resources. For instance, a provider may offer an availability SLA (e.g., 99.9%) on a group of VMs, i.e., SLA is violated if aggregate uptime of all VMs falls below 99.9%. A group SLA is more flexible than a per resource SLA because it leaves provider the wiggle room for repeatedly terminating or migrating resource heavy instances while still meeting the group SLAs. However, repeatedly terminating the same instance may annoy the customer.

4. THEORETICAL BASIS FOR OVERSUBSCRIPTION PROBLEM

The oversubscription problem can be considered a variant of online multiple constraint multiple knapsack problem. In the classical single constraint single knapsack problem (SCSKP), the objective is to maximize profits of items in a knapsack while adhering to a single constraint, i.e., knapsack size. In multiple constraint, single knapsack problem (MCSKP), the number of constraints is more than one, such as size and weight, but the objective is to maximize profit. For a physical machine, the constraints will be the maximum memory, CPU, and disk utilization. In multiple constraint multiple knapsack problem (MCMKP), the objective is to maximize profit of items in all knapsacks while adhering to knapsack constraints. MCMKP is an NP-hard problem.

The oversubscription problem and methods for overload remediation such as quiescing a VM or live migration impose several twists on the MCMKP problem. First, oversubscription is an online version of MCMKP. The items in a knapsack can grow and shrink, which correspond to VMs using and releasing resources of a PM. When a constraint is violated, the scheduler of online MCMKP must take actions for overload remediation using VM quiescing or live migration. Each action has a corresponding cost. As long as the cumulative cost of these actions is below the thresholds specified in SLA, the actions can be taken without any penalty.

More formally, consider a data center, comprising of M physical machines (PMs) running an aggregate of V virtual machines (VMs), where each physical machine is overcommitted by a factor of *oc*. For instance, when $oc = 2$, the aggregate provisioned resources of VMs (e.g., CPU and memory) running on a PM are twice that of the PMs resources. The overcommit factor can also be different for every PM and is the subject of exploration. For simplicity of analysis, consider that the provider runs identically provisioned VMs on PMs and charges the same cost per VM to the user of the VMs. However, each user may run a workload which may not always use the resources allocated to the VMs, thereby providing an opportunity to oversubscribe the underlying PM.

Let U_{i,k_i} denote the utility function of the k_i^{th} VM (v_{i,k_i}) running on i^{th} PM. It indicates whether the VM's performance meets the SLA requirement as promised by the provider to the user of v_i. We define the normalized range of the utility function as between $[-1, 1]$. A negative utility captures the impact of potential SLA violations on the profits of cloud provider. Since a VM can have multiple resources $r \in$ (CPU, memory, disk, network), we model the utility for each resource, r, on a VM as U^r_{i,k_i}. In that case, the total utility across all resources U_{i,k_i} for v_{i,k_i} is as follows:

$$U_{i,k_i} : \mathbb{R} \rightarrow [-1, 1] \qquad U_{i,k_i} = \min_r U^r_{i,k_i} \tag{1}$$

Here, we take the minimum value across all resources, instead of their sum, because a single overloaded resource (e.g., memory swapping) can significantly impact the performance of the running VM, easily causing an SLA violation.

Let $v_{i,1}, \ldots, v_{i,k_i}$ be the VMs that are assigned to a PM m_i and $v_{j,1}, \ldots, v_{j,k_j}$ be the VMs assigned to a PM m_j. k_i and k_j denote the total number of VMs run on PM i and j, respectively. Let $R_{i,1}, \ldots, R_{i,r_i}$ denote the resource consumption of PM i. The assignment of VMs to PMs and load on each VM and PM constitutes the state space A of an oversubscribed data center. Specifically, state tuple a for VMs running on PM i is defined as (2)-(4):

$$(v_{i,1} \rightarrow m_i, \ldots, v_{i,k_i} \rightarrow m_i) \tag{2}$$

$$(R_{((v_{i,1}),1)}, \ldots, R_{((v_{i,1}),r)}, \ldots, R_{((v_{i,k_i}),1)}, \ldots, R_{((v_{i,k_i}),r)}) \tag{3}$$

$$(R_{i,1} \ldots R_{i,r_i}) \tag{4}$$

where (2) denotes the assignment of VMs to PM i, $R_{((v_{i,1}),r)}$ denotes the resource consumption of r_{th} resource on VM $v_{i,1}$ in (3), and $R_{i,r}$ denotes the resource consumption of r_{th} resource on PM i in (4). The total utility of all VMs in this state a is the sum total of the utility of each VM, i.e., $U_a = \sum_i \sum_{k_i} U_{i,k_i}$.

Assume a central scheduler similar to VMware's DRS [7] is configured with the overload mitigation policies and continuously monitors the performance of VMs. If a VM starts suffering from overload due to oversubscription, the scheduler kicks in overload mitigation using one or more mechanisms discussed in Section 3.1. A key issue in those mechanisms is the selection of VMs for quiescing and live migrations while meeting placement constraints, which can be according to a configured policy.

In one policy which we refer to as 'min', the VMs with the smallest resource footprint can be selected as candidates for live migration. For example, VM with the smallest consumed memory is considered a candidate for live migration. In the greedy or 'max' policy, the VM with largest resource footprint is selected for live migration. If live migration may not be possible because other 'near by' PMs are also overloaded, or if live migration is prohibitively expensive due to lack of shared storage, VMs can be selected for quiescing according to a similar policy, such as minimum or maximum resource usage, or fairness. Our framework allows comparing of different VM selection strategies for mitigating overload that take advantage of VM quiescing, live migration, or combination of both.

Let P_A denote the steady state probability matrix which indicates the probability of a system being in state a. Let

λ_Q and λ_M be the rate of state transitions due to quiescing and migration. λ_Q and λ_M are determined according to the workload profile of VMs and the configured overload mitigation policies which must kick in to satisfy the VM's SLA requirement and the corresponding utility function. Let O_Q and O_M denote the overhead functions for quiescing and migration, respectively, which have the same range as the utility function. Then, the total utility of the system is given as:

$$U_A = \sum_A P_a(U_a - (\lambda_Q O_Q + \lambda_M O_M)) \quad (5)$$

The formulation states that the aggregate utility of the system is the utility of being in state a minus the rate of stealing and migration (and the associated overhead) into that state, times the probability of being in state a, summed over all states. The formulation is useful both during resource planning in cloud and live cloud operation. For a given workload distribution, a provider can iteratively use this formulation to determine the maximum number of VMs that can be run while meeting the SLAs. Similarly, for a given overcommit factor (e.g., 2), a provider can determine the maximum VM resource request rate derived from known distributions. Conversely, a provider can check for different overcommit factors for different machines. In all these scenarios, a provider can evaluate policies for mitigating overload that use a combination of VM quiescing or live migration. To keep the state space of model tractable, one can only consider equation (3) in utility calculations, albeit, at some loss of accuracy.

4.1 Objective Function

Recall our assumption, that the provider configures all VMs identically and charges same cost per VM to the user, but the workload profile of each VM may be different. Thus, by maximizing the utility function U_A defined in (5), a provider maximizes its profit. Specifically, the number of VMs V for which the utility function is maximized, i.e.,

$$\arg\max_V U_A$$

5. SIMULATION AND RESULTS

We wrote an event drive simulator to develop an understanding of oversubscription in cloud. Our simulation setup comprises of 40 PMs, each configured with 64 GB of RAM. Each PM is oversubscribed by a factor of two, i.e., the total provisioned memory of VMs on a PM is twice the size of physical host memory of 64 GB. In our simulation, the number of VMs do not change, but the memory request rate on each VM varies according to a known distribution. As a first step, we only consider overload of physical memory; other resources such as CPU, disk, and network are being considered in the ongoing work. The reason we choose physical memory overload over say CPU overload was that applications cannot make any progress when suffering memory overload. In contrast, in CPU overload, the applications may still be able to progress, albeit at a reduced available processing power. The simulation was run for 30 days of simulated time.

The memory request interarrival rate is exponentially distributed. Every interarrival time, the size of the request is chosen from a probability distribution. In this paper, we experiment with exponential and pareto distributions for memory request sizes, but omit the pareto results due to lack of space. The choice of these distributions is motivated by our effort to bring insights into cloud oversubscription and by our observation of VM memory requests in production data centers. In the future, we plan to incorporate traces from real VM workloads.

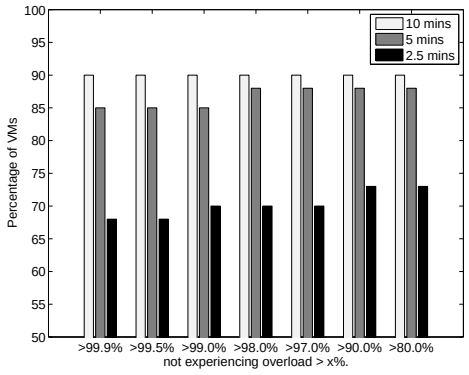

Figure 2: Relationship between overload detection interval (fixed at 5 minutes) and request interarrival time on VM (varied between 10 minutes, 5 minutes, and 2.5 minutes).

The choice of keeping VMs and overcommit factor constant but varying the memory requests on VM was motivated by our desire to keep simulation tractable and to bring forth useful insights. The converse is also possible, i.e., varying the VM interarrival time and overcommit factor, but keeping the memory request rate constant will yield similar insights.

5.1 Defining Overload

We consider overload to have occurred if a physical host experiences memory pressure above a threshold for more than five minutes. The choice of five minutes is motivated by Amazon EC2 SLA described in Section 3.3, which states that a VM is considered unavailable if a customer cannot access it for five contiguous minutes or launch replacement instances within five contiguous minutes. Our overload detection threshold is set to 95% of the provisioned PM memory or 60.8 GB. We assume that during period of memory overload, a customer is unable to access its instances.

5.2 Relationship Between Overload Detection Interval and Request Interarrival Time

As our first experiment, we explore the relationship between overload detection threshold and VM memory request interarrival interval, both of which are derived from exponential distributions. We vary the interarrival rate so that the ratio of memory detection threshold time period to average interarrival interval is 2, 1, and 0.5, respectively. Specifically, the overload detection interval is kept constant at five minutes, where as the interarrival time between memory request is varied between 2.5 minutes, 5 minutes, and 10 minutes. The overcommit factor was two, and each PM ran 64 VMs, and each VM was configured with provisioned memory of 2 GB. Thus, a total of 40 physical machines were running 2,560 VMs. The mean memory request size on each VM was 32.5% of the provisioned VM memory (or 650 MB)

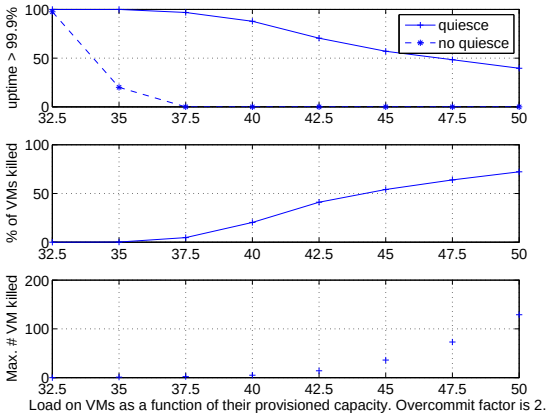

Figure 3: (Top) Performance of quiesce scheme for mitigating overload vs. do-nothing approach. (Middle) Percentage of VMs terminated. (Bottom) Maximum number of times any of the 2,560 VMs was shutdown.

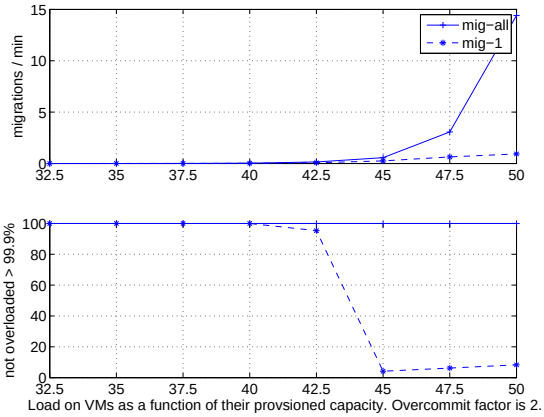

Figure 4: Performance of a live migration scheme, which migrates VMs until overload is relieved (mig-all) and which can at most migrate one VM per minute (mig-1).

and was exponentially distributed. Therefore, the total average memory consumption on a PM running 64 VMs was 41.6 GB. If the mean request size on each VM was 50% of provisioned memory, then the total average memory consumption on the physical host will be 64 GB or equal to its maximum capacity. However, since the request size is capped at the maximum provisioned VM memory, the average PM memory consumption is slightly lower than expected.

Figure 2 shows the result. The x-axis represents VMs not experiencing overload more than x% of the time, and y-axis shows the percentage of VMs. Consider the left most bar in the graph, which indicates that close to 90% of 2,560 VMs do not experience overload more than 99.9% of the time. The figure clearly shows a drop by a factor of 1.4 in percentage of VMs not experiencing overload, when interarrival time reduces from 10 minutes to 2.5 minutes. Recall, that overload detection interval was fixed at five minutes. The figure clearly shows that overload detection time must be smaller or close to resource request interarrival times. Similar results were obtained when request size and interarrival time were pareto distributed.

5.3 Mitigating Overload By Quiescing VMs

When a PM's memory is overloaded, all VMs running on the PM are potentially affected. As discussed in Section 3.1, one way to relieve overload on a PM is to quiesce or shutdown one or several VMs. The terminated VMs can eventually be restarted when the PM is no longer overloaded. The key question is how much is the gain in VMs' SLAs (non-overloaded uptime) when quiescing is used compared with a do-nothing approach? To answer this question, we use the same simulation setup as described in Section 5.2. We implement the VM quiescing scheme on a PM as follows. If a PM is overloaded for more than one half of overload detection threshold (5 minutes in our setting), we select the VM among all running VMs with the most memory used and terminate it, thus following a 'max' strategy. Our algorithm repeatedly terminates VMs till overload is relieved. Once the overload has been relieved, our algorithm at least waits for a full minute before restarting the terminated VMs. The algorithm restarts the VM that has been shutdown for the longest duration. The algorithm does not immediately restart all the VMs, but rather restarts one VM, every time any VM makes a new memory request.

Figure 3 [top] shows the percentage of non-overloaded VMs with uptime $> 99.9\%$ as load on each VM is increased. The x-axis represents the mean request size per VM as a function of its provisioned memory size, while y-axis represent the percentage of VMs. The results clearly show that quiescing VMs to relieve overload can greatly improve the percentage of running VMs not experiencing overload (approximately, by a factor of 2) when compared with a do-nothing approach.

Another interesting question is what is the maximum number of times any VM is shutdown during entire simulation run of 30 days? Since the arrival request size is exponentially distributed, it comes as no suprise that maximum number of times a VM is shutdown is small (< 5), when percentage of VMs with uptime greater than 99.9% is more than 80%. However, if few VMs were to be heavy hitters in terms of memory requests, the 'max' scheme will repeatedly quiesce those VMs, which may not be a desirable outcome from the perspective of that VM's user.

Figure 3 [middle] shows the number of VMs that were terminated one or more time to mitigate overload as mean request per VM increases. The top and middle figures show a close co-relation between percentage of VMs with uptime $> 99.9\%$ and percentage of VMs killed one or more time.

5.4 Mitigating Overload By Live Migration

Using the setup used in earlier section, we performed simulations to determine if live migration can alleviate overload without terminating VMs. We implement the live migration scheme on a PM as follows. If a PM is overloaded for more than one half of overload detection threshold (5 minutes in our setting), we select the VM among all running VMs with

the most memory used and migrate it to a PM with the most spare memory. Our algorithm 'mig-all' repeatedly migrates VMs till overload is relieved. To understand the maximum gains possible using live migration, we assume the cost of live migration to be zero. Figure 4 plots the migrations per minute and percentage of VMs not experiencing overload. The x-axis represents the mean request size per VM as a function of its provisioned memory size. The figure shows that the number of VM migrations per minute is 14, when mean request size per VM is 50% of its provisioned memory size. Recall that overcommit factor was two. Live migration involves migrating the live memory foot print of a 2 GB provisioning VM memory. Given the mean request size per VM is 50% of provisioned memory, the live memory footprint is 1 GB on average. It is unrealistic to expect that more than one VM per minute can be live migrated to another PM when each PM is connected to 1 Gb/s Ethernet.

We implemented a scheme 'mig-1' which migrates at most one VM per minute to alleviate overload. Figure 4 shows the performance of this scheme. Although, the number of VMs being migrated per minute never exceeds one, PMs start to experience overload at much lower memory request sizes. The figure clearly shows that simply using live migration may not be sufficient to alleviate overload, and that a combination of VM quiescing and live migration schemes may be needed. We are exploring this combined scheme as part of ongoing work.

6. ONGOING WORK

The simulations help us develop an understanding of oversubscription in cloud, but several questions remain. We list some of these questions below.

- To what extent a combination of VM quiescing and live migration schemes perform better than the individual schemes?
- Does asymmetry in oversubscription levels across PMs (within the same rack) and workload distributions lead to a higher overcommit level?
- When identical or asymmetric capacity VMs have different SLAs, which overload mitigation scheme gives the best results?
- When the available SLAs are defined per VM group instead of per VM, can it be leveraged to improve the performance of underlying overload mitigation scheme?
- How are the results affected when other resources such as CPU, network, and disk are oversubscribed?
- How can we answer all of the above questions for real workloads in a test-bed or deployed environment?

7. RELATED WORK

The related work for modeling can be divided into the systems and theoretical work.

Urgoankar *et al.* [6] were the first to show the feasibility and benefits of oversubscrition for running different components of heterogeneous applications in a non-virtualized multiple physical machine environment. They demonstrated oversubscription of CPU and network resources of physical machines. Tang *et al.* [5] considered an application placement controller for non-virtualized environments. Sand Piper [11] work devised black box and gray box strategies for VM migration. However, they did not consider an oversubscribed environment and did not compare the cost of migration with that of VM quiescing. Further, their hotspot mitigation scheme was not directly tied to a concrete SLA.

8. CONCLUSION

We have developed a framework for understanding oversubscription in a cloud environment. We presented overload mitigation mechanisms and service level agreements, which must be coalesced together in any overload mitigation algorithm. We then described the theoretical basis of the oversubscription problem, and presented results from simulations which evaluate the relationship between overload detection interval and request interarrival time, and evaluated the performance of an overload mitigation scheme which terminates VMs and performs live migration for relieving overload. As part of ongoing work, we are exploring the impact of a combination of VM quiescing and live migration schemes, asymmetry in oversubscription levels and SLAs, and group SLAs.

9. REFERENCES

[1] Amazon EC2. `https://aws.amazon.com/ec2/`, 2011. [Online; accessed January 2012].

[2] Amazon EC2 SLA. `https://aws.amazon.com/ec2-sla/`, 2011. [Online; accessed August 2011].

[3] Rackspace Cloud Servers SLA. `http://www.rackspace.com/cloud/legal/sla/`, 2011. [Online; accessed August 2011].

[4] C. Tang. Fvd: a high-performance virtual machine image format for cloud. In *Proceedings of the 2011 USENIX conference on USENIX annual technical conference*, USENIXATC'11, pages 18–18, Berkeley, CA, USA, 2011. USENIX Association.

[5] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici. A scalable application placement controller for enterprise data centers. In *Proc. of WWW*, 2007.

[6] B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource overbooking and application profiling in shared hosting platforms. In *Proc. of OSDI*, Boston, MA, Dec. 2002.

[7] VMware. Distributed Resource Scheduler. `http://www.vmware.com/products/drs/overview.html`, 2011. [Online; accessed April 2011].

[8] C. A. Waldspurger. Memory resource management in VMware ESX server. In *Proc. of USENIX OSDI*, Boston, MA, Dec. 2002.

[9] D. Williams, H. Weatherspoon, H. Jamjoom, and Y.-H. Liu. Overdriver: Handling memory overload in an oversubscribed cloud. In *Proc. of VEE*, March 2011.

[10] Windows Azure Compute SLA. `https://www.microsoft.com/download/en/details.aspx?displaylang=en&id=24434`, 2011. [Online; accessed August 2011].

[11] T. Wood, P. Shenoy, and A. Venkataramani. Black-box and gray-box strategies for virtual machine migration. In *Proc. of USENIX NSDI*, Cambridge, MA, Apr. 2007.