Multiplexing Low and High QoS Workloads in Virtual Environments

Sam Verboven, Kurt Vanmechelen, and Jan Broeckhove

University of Antwerp,
Department of Computer Science and Mathematics,
Middelheimlaan 1, 2020 Antwerp, Belgium
sam.verboven@ua.ac.be

Abstract. Virtualization technology has introduced new ways for managing IT infrastructure. The flexible deployment of applications through self-contained virtual machine images has removed the barriers for multiplexing, suspending and migrating applications with their entire execution environment, allowing for a more efficient use of the infrastructure. These developments have given rise to an important challenge regarding the optimal scheduling of virtual machine workloads. In this paper, we specifically address the VM scheduling problem in which workloads that require guaranteed levels of CPU performance are mixed with workloads that do not require such guarantees. We introduce a framework to analyze this scheduling problem and evaluate to what extent such mixed service delivery is beneficial for a provider of virtualized IT infrastructure. Traditionally providers offer IT resources under a guaranteed and fixed performance profile, which can lead to underutilization. The findings of our simulation study show that through proper tuning of a limited set of parameters, the proposed scheduling algorithm allows for a significant increase in utilization without sacrificing on performance dependability.

Key words: Workload multiplexing, Virtualization, Overbooking, Scheduling

1 Introduction

A current trend in IT infrastructure management is the reliance on virtualization technology to mitigate the costs of application and IT infrastructure deployment, management and procurement. Virtualization technology allows one to manage an application and its execution environment as a single entity, a *virtual machine* (VM). It allows for the full configuration of an application and its execution environment to be captured in a single file, a virtual machine *image*. These virtual machine images can be deployed in a hardware-agnostic manner on any hardware that hosts a compatible virtual machine *monitor* (VMM) such as Xen [1] or VMware's VMM. VM monitors thereby offer flexibility in partitioning the underlying hardware resources and ensure isolation between the different virtual machines that are running on the same hardware. Aside from the benefits of this technology in the context of privately owned data centers, these features have also fostered the development of a new IT infrastructure delivery paradigm that is based on outsourcing. The possibility to deploy an entire environment in a low-cost and hardware-neutral manner has paved the way for *cloud* [2, 3] infrastructure as a service (IaaS) providers to open up their large datacenters to consumers, thereby exploiting significant economies of scale.

One of the most well known providers in this new market is Amazon with their Elastic Compute Cloud (EC2) [4] offering. As is typical for an IaaS provider, Amazon offers a discrete number of resource types or *instance types* as they are called, with varying performance characteristics. Such an instance type delivers a guaranteed level of compute capacity. An EC2 *small* instance type for example, contractually delivers a performance that is equivalent to a 2007 Opteron processor with a 1.0-1.2 GHz clock frequency. The performance guarantees in this service delivery model are crucial because the use of the compute service is paid for by the hour, and not by actual compute capacity delivered or used. The combination of these performance guarantees and the fact that virtual machine workloads can vary significantly can lead to infrastructure underutilization in absence of corrective measures. In addition, the ability to buy *reserved* instances at EC2 that have a guaranteed level of performance *and* availability, further increases the chances for underutilization. The recent addition of a spot market [5] for EC2 instances whereby instance types are dynamically priced and potentially killed by the provider if their standing bid does not meet the spot price, provides an indication for this problem of (temporary) underutilization. The addition of this market mechanism changes the scheduling problem within the datacenter from one in which a given set of workloads need to be balanced out over the available hardware, to one in which the change of an admission parameter,the instance's spot price, can trigger an influx of additional VM workloads into the datacenter. These workloads run under lower availability guarantees as they can be shutdown by the provider if they cause interference with workloads that run under a high availability regime, such as the reserved instance or on-demand instances at EC2[1].

Scheduling workloads that have low priority and quality of service (QoS) guarantees in terms of performance, alongside with high-QoS workloads thus offers a possibility to deal with underutilization. Consider for example the addition of a batch job workload to a 4-way server that is running a VM with four cores hosting a high priority web service. The web service's spiky load pattern opens up the possibility for filling in underutilized periods with the batch workload. Such a scheduling approach must ensure that high-QoS workloads do not suffer from performance degradation caused by their multiplexing with low-QoS workloads. At the same time, enough low-QoS workloads should pass admission

[1] Note that EC2 uses an indirect mechanism for this by increasing the spot price to a level that rises above the standing bid of an adequately high number of spot instance workloads. This clears them for shutdown under the contractual rules of the trading agreement.

control in order to achieve the highest possible utilization and throughput of the infrastructure.

Although some commercial products exist, such as VMware's vSphere, that perform load balancing in a cluster for a given set of virtual machines, no definite solution exists today for tackling this problem if a free decision can be made to accept additional low-QoS workloads. In this paper, we present a simulation framework to analyze the performance of VM scheduling problems and evaluate a scheduling algorithm that is tailored towards the multiplexing of these high- and low-QoS workloads in a virtual machine context. We demonstrate that by tuning a limited set of parameters a tradeoff can be made between maximizing utilization and avoiding workload interference.

2 Model

2.1 Resource and Job Model

In this contribution, we research the VM scheduling problem within the context of the following model. We explore the problem in a setting with one infrastructure provider P, that hosts a set of m machines M_j $(j = 1, \cdots, m)$. These are considered to be identical parallel machines so each machine is able to execute any job from the set of n jobs J_i $(i = 1, \cdots, n)$, and for the machine's processing capacity s_j we have, $\forall i, j \in \{1, \cdots, m\} : s_i = s_j = 1$. A job, which models the execution of a virtual machine instance, has a varying load pattern over time and is sequential, i.e. it runs on only one machine at a time. A job has a release time r_i, and a duration p_i. We consider two types of QoS levels for jobs. High-QoS jobs must be able to start at time r_i and should be able to allocate the full processing power of the machine on which they are deployed. These jobs are not preemptible, e.g. a virtual machine running a relational database. Low-QoS jobs can be preempted at a fixed cost c_p. In this work, we assume that job preemption requires a suspension of the virtual machine. Equivalently, a resumption of a virtual machine instigates a cost c_r. The job startup costs (c_b) and termination costs (c_t) are also modeled as we are dealing with VMs. For preemption, we only consider the case wherein a VM is swapped out of memory to make room for the other VMs that run on the server. An example of a workload that is amenable to a low-QoS regime is a virtual machine that executes low-priority batch jobs.

A machine corresponds to a virtualized core of a server that runs a virtual machine monitor. The provider P operates a cluster of such servers. A machine can accommodate more than one job at a time. We assume that the distribution and multiplexing of a VMs workload over the virtual cores of a server is managed by the virtual machine monitor and do not explicitly model this behavior. We also do not model the overheads that such multiplexing brings in terms of technical considerations such as I/O contention for resources or cache line invalidations. Although these aspects can certainly have a significant impact on this study, they are also very application dependent and difficult to model and simulate. In that respect, this study maps out the maximum performance that can be attained under the proposed scheduling approach.

2.2 VM management model and simulation framework

For managing the distribution of virtual machines over multiple servers in the cluster a *virtual infrastructure manager* (VIM) is required. There are multiple such managers currently available such as vSphere (VMWare's commercial offering), or one of the open source alternatives such as OpenNebula [6] or Eucalyptus [7]. Depending on the capabilities of the VIM, a set of features and operations is available to manage the execution of the VM instances on the cluster. Because of its generality, we have chosen to model our scheduling problem in the context of the features offered by the OpenNebula toolkit. The open nature of the project, the emphasis on being a research platform and the generality of its feature set are the main factors that influenced this choice.

One of the schedulers already available for OpenNebula is the Haizea [8, 6] scheduler. The VM operations available to the scheduler are *shutdown*, *start*, *suspend* and *resume*. The scheduler is assumed to have no knowledge of p_i. In order to deal with infrastructure underutilization, we take an overbooking approach. That is, we allow the scheduler to allocate more resources than physically available on the cluster node. Such an overbooking has to be actively managed by active scheduling decisions in order to limit the interference of low-QoS loads with high-QoS loads. As the Haizea scheduler already supports many of the features required for overbooking, such as the support for differentiation between multiple job types, it is chosen as the basis for our scheduler.

All of the scheduler's decisions result in a series of commands and corresponding VM states that can be used to drive the two enactment backends available in Haizea. The first is a simulated backend used in the presented experiments, the second drives the OpenNebula virtual infrastructure engine where Haizea can be used as an alternative to the default scheduler. One of the major benefits of the second backend is that all the scheduling algorithms implemented within the extended framework are automatically compatible with OpenNebula. An advantage of this choice is that the results of our simulation studies can be verified in a real-world setting without much additional cost.

Haizea's simulation mode uses a simulation core that keeps track of all *actions* that are scheduled with a specific firing and finishing time. The simulation steps through time by subsequently adjusting the simulator's virtual clock to the time of the next action. At each step, the state of the simulated environment is updated and user code can step in to schedule new actions. A single VM operation, such as suspend, can involve one or more actions, depending on the level of detail in the VM management model. For example, one could explicitly model the time required for state checkpointing, or the I/O operation involved in storing the checkpoint.

With a configurable time frequency, our scheduler performs an *overbooking* step. In such a step all available machines are polled to obtain the active jobs and their current utilization. This information is then used to determine all the VM operations that are required, based on the scheduling policy's options. Interspersed with these fixed steps lie *management* steps. During the management steps all events that do not coincide with overbooking step times are performed e.g. issuing a shutdown command when a VM has finished its workload.

3 Scheduling Algorithm

Any overbooking scheme will have the same general goal: reduce resource wastage due to underutilization while at the same time having a minimal impact on the existing resource users. As a result of their suspend and resume capabilities VMs are uniquely suited for this goal provided they have different types of QoS requirements. A lower priority VM can be suspended and resumed at a later, more opportune time and/or location without losing any performed work. The scheduler determines the suitability of machines for low-QoS jobs and only launches the job if sufficient resources are available. For high QoS jobs, the scheduler installs reservations to make sure resources are available for the entire duration of the workload. Low-QoS jobs are queued up until machines are available.

As jobs only use a single machine, the amount of jobs supported by a single cluster node can be expressed in *slots*. Each slot is equivalent to the processing capability of a single CPU core. As such, we will refer to a machine M_i as a slot in the remainder of this paper. Slots provide a convenient abstraction to specify both the available physical resources as well as the maximum allowed amount of overbooking.

High-QoS jobs may require the full processing capacity of the reserved slots at some point in time but it is reasonable to assume this is not permanently the case. The reserved but unused resources pose both an opportunity and a challenge. There is an opportunity to increase overall utilization by scheduling in low-QoS workloads. Depending on the QoS guarantees, interference with high-QoS workloads with must be completely avoided or kept within reasonable bounds. In contrast to the EC2 approach, we want to preserve the work that has been completed in a low-QoS VM and therefore do not kill it if it is detected to interfere with high-QoS VMs. Therefore, our scheduler must take into account the overheads of suspending and resuming low-QoS VMs. Suspending as well as starting and stopping a VM can be a resource intensive operation. Depending on the configuration of the cluster, it is possible that all four major resources (CPU, memory, disk and network) are heavily taxed.

We quantify the interference between VMs by measuring the CPU utilization on a node in excess of 100%. As mentioned before, this is only one dimension of interference that can exist between VMs that are deployed on the same node. Other dimensions such as contention for disk I/O bandwidth will be investigated in future work.

A simple and effective method to put restrictions on the allowed ranges for overbooking is the introductions of bounds. The base algorithm determines its actions using a lower and an upper bound. The lower bound puts a limit on the maximum node utilization for nodes where new low QoS VMs are booted. The upper bound is used to decide when a VM should start suspending. Keeping in mind the overhead of starting and suspending a VM, the algorithm will not schedule more than one of these operations simultaneously one a node.

Our scheduling algorithm works in two steps: scheduling new overbooking requests and evaluating running requests. The first step, for which pseudo-code is shown in Algorithm 1, works as follows. The algorithm starts by obtaining a list of all the nodes that can currently support an extra VM. The suitability of a node is determined by comparing the node utilization (including the loads introduced by possible overbooked VMs) with a configurable lower bound. All nodes with a utilization lower or equal to this lower bound are added to a list of overbooking candidates. After suitable candidates are found the list of low-QoS requests is updated: incoming requests are added to the back of the queue while suspended requests are added to the front. Suspended requests are ordered by initial arrival time with the oldest appearing at the front of the queue. With all necessary data gathered, VMs can be scheduled until either the available nodes or requests are exhausted.

```
Input: Set of nodes, Set of vm_requests, lower_bound
foreach Node i do
    if Utilization(i) $\leq$ lower_bound then
        available_nodes.add(i) ;
    end
end
Update(vm_requests) ;
while available_nodes remaining & vm_requests remaining do
    vm = vm_requests.pop() ;
    n = available_nodes.pop() ;
    Schedule(vm on Node n) ;
end
```

Algorithm 1: Adding Overbooked VMs

Since utilization is a volatile property the conditions for overbooking will need to be evaluated at regular intervals. The pseudo code for this part of the algorithm can be found in 2. All nodes supporting one or more overbooked VMs are evaluated, and if the total utilization equals or surpasses the set lower bound the VM that was added last will be suspended.

```
Input: Set of nodes, upper_bound
foreach Node i do
    if Utilization(i) $\geq$ upper_bound then
        vm = overbooked_vms(i).get_last() ;
        Suspend(vm) ;
    end
end
```

Algorithm 2: Suspending Overbooked VMs

4 Experiments

In this section, we evaluate the performance of our scheduling algorithm. We first outline our experimental setup after which we present and discuss our results.

4.1 Experimental setup

Our experimental setup consists of three major aspects: the cluster used to deploy the VMs, a list of high- and low-QoS requests and the load generators attached to the requests. The cluster consists of 50 homogeneous octacore nodes. To generate a non-trivial synthetic load pattern that is reminiscent of the behavior of real-world workloads, we introduce the following three different application types[2] following [9]:

Noisy: Starting from a mean utilization value μ, a load pattern is generated by drawing random numbers from a normal distribution $N(\mu, 15)$. An example of a noisy load pattern for $\mu = 75$ can be found in Figure 1.

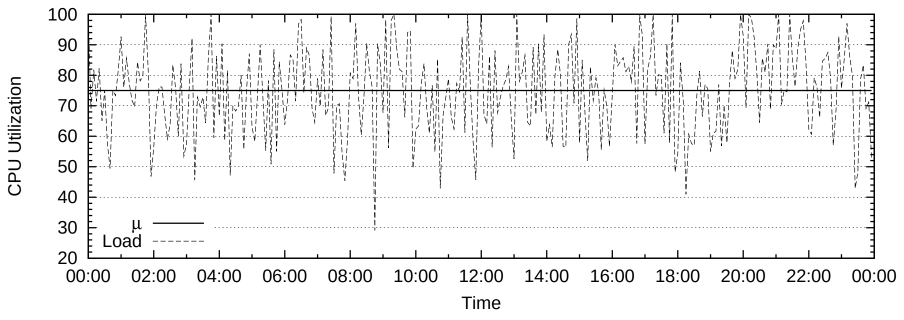

Fig. 1. Sample noisy load pattern.

Spiky: This load pattern is based on a normal distribution with $\sigma = 5$. To add load spikes to the pattern, each drawing of the load distribution has 1% chance of generating a spike with 90% chance of having a positive one. Each spike has 50% chance of continuing. An example spiky load pattern for $\mu = 75$ can be found in Figure 2.

Business: A business load pattern is slightly more complicated in that a function is used to determine the μ parameter of the normal distribution $N(\mu, 5)$ depending on the time of day. The value of μ is calculated with a piecewise function that represents utilization fluctuations coinciding with business hours. The function is configured with a minimum (*min*) and a maximum (*max*) utilization value. Utilization rises from *min* to *max* between 8.00 and

[2] By manipulating a limited number of parameters we can emulate a wide range of applications.

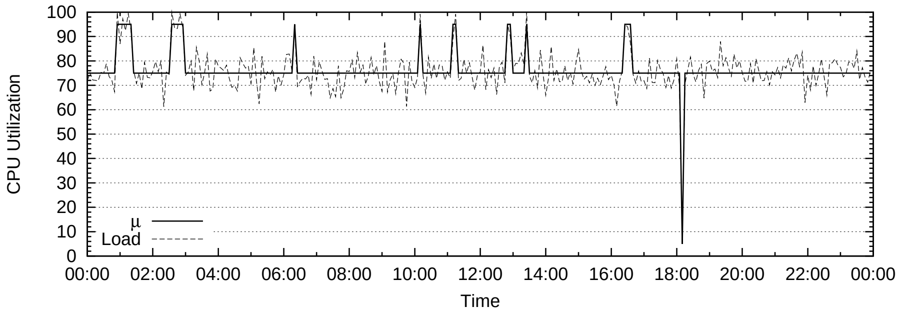

Fig. 2. Sample spiky load pattern.

10.00 in the morning. Between 11.30 and 13.30 there is a slight drop representing lunch hours. In the evening there is a second decline dropping back to *min* between 16.00 and 18.00. The incremental utilization changes between *min* and *max* are calculated by adjusting the amplitude and period of a *sinus* function. During weekends, the function returns the minimum value. An example business load pattern for $min = 50$ and $max = 90$ is shown in Figure 3.

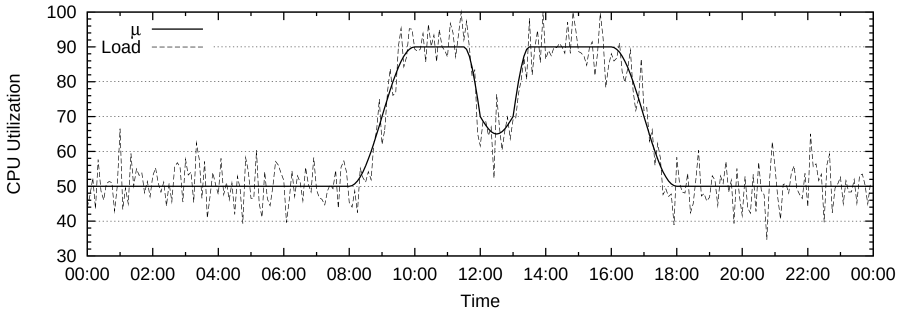

Fig. 3. Sample business load pattern on a weekday.

Each high-QoS application has an equal chance of generating one of the three load patterns. For the spiky and noisy load patterns, μ is drawn from a normal distribution $N(75, 15)$. For the business load pattern, $min = 50$ and $max = 90$. An example of a possible workload during a weekday on a node in the cluster can be found in Figure 4. High-QoS applications are generated in such a manner that load patterns are randomly scheduled among the different nodes on the cluster. Each low-QoS application has a noisy load pattern with $\mu = 90$ simulating CPU intensive batch jobs. Each separate application consists of a single job. High-QoS jobs are generated in such a manner that all physical slots are continuously occupied. Using 50 octacore nodes this means there are 400 high-QoS applications running at any given time, one for each core. The low-QoS job arrival rate is set a level that ensures the queue never becomes empty. The maximum amount of concurrently executing low-QoS applications depends on the overbooking slots per node.

All application runtimes are generated according to a geometrical distribution. If X is the runtime in minutes, the probability is expressed in equation 1 for $n = 30, 60, 90, ...$ with p equaling 0.1% and 1% for respectively high- and low-QoS applications.

$$Pr\left[X = n\right] = p\left(1-p\right)^{\left(\frac{n}{30}-1\right)} \tag{1}$$

Preliminary tests indicated that the results for running the simulation for one week and for one month produced equivalent results. This is a logical consequence of the weekly repeating pattern. To reduce the time needed to produce results for the numerous tests, we reduced the time horizon of the simulation to one week. The frequency for running the overbooking logic was set to 5 minutes. The costs for VM operations were configured as $c_b = c_p = c_r = c_t = 30s$. Providing an estimate for VM operations in a cluster environment depends highly on not only the storage and network configuration but also on the target VM memory usage. The 30s estimate should be viewed in the context of a cluster using fast network storage to provide the VM images and VM instances using 1 GB of memory. This assumption removes the need to model migration overhead when resuming VMs on different nodes.

Executing the scenario without overbooking logic results in a mean CPU utilization of 69.4% during a total of 67,200 workload hours. Every test consist of three parameters: available overbooking slots, upper- and lower bound. These are chosen in function of the relatively high average utilization on the simulated cluster. The amount of overbooking slots was taken to either be 1, 2 or 3. We varied the upper and lower bounds in increments of 5 between [85, 95] and [60,80] respectively. Since each CPU core can maximally account for a utilization of 12,5%, the minimum difference between lower and upper bound is taken to be 15%. Relaxing this constraint will often result in immediately suspending the VM once it becomes active. All other lower bounds are set 5% apart going down until 60.

4.2 Results

The outcome of the experiments is gathered into Tables 1-3, each containing the results of the test performed for a set amount of overbooking slots. The first column contains upper and lower bounds. The third column shows the average utilization achieved when the overbooking logic is active. The average utilization of 69.4% achieved without overbooking, increases to more than 87% for the scenario with three slots, a lower bound of 80 and upper bound of 95. A conservative bound configuration of 85-60 using a single overbooking slot, leads to

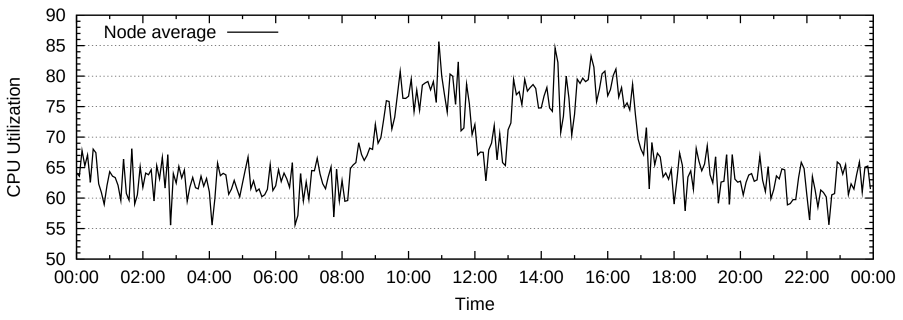

Fig. 4. Sample load pattern on a single eight core node during a weekday.

a utilization of 73.7%. The fourth column contains the hours of workload that have executed within overbooked low-QoS VMs. This total does not include any VM operation overhead, only active VMs can contribute to the total. The fifth column shows the amount of VM suspensions. The second to last column contains the amount of *degradation points* if the results are interpreted without any overhead. Degradation points are all overbooking time steps where a total load was recorded that would impact the high-QoS VMs. The last column contains the amount of degradation points taking into account a 5% overhead.

From the preliminary results in Tables 1, 2 and 3 we can reach some initial conclusions with respect to the parameter variations and their results. For the purpose of this discussion we will refer to the difference between upper and lower bounds as the overbooking *window size*. *Negative effects* are considered to be a combination of increased suspensions (and the resulting resumptions) and an increase in the amount of degradation points at both 95 and 100%.

We will first look at the impact caused by the amount of available overbooking slots. The influence of the amount of overbooking slots is lowest in the scenarios with the lowest bound values. This can be attributed to the fact that the average utilization without overbooking is already relatively high, and only allows for a single overbooked VM when bounds are set low. When the bounds are increased more interesting results can be observed. Moving from one overbooking slot to two yields higher utilization levels and often lower negative effects for similar bound values. A single overbooking slot performs only slightly better when the lower bound is set at 60 and total utilization is lowest. Increasing the slot amount to a maximum of three overbooked VMs on the other hand results in similar utilization levels while having the same or more negative effects. A higher maximum increase in utilization can be achieved but there is a substantial increase in the amount of negative effects as well. It seems that in most cases, two overbooking slots is the most appropriate setting for this type of workload.

A detailed side by side comparison of Tables 1-3 shows that although the numbers may vary, the trends that can be detected are similar. There are two trends that deserve some further discussion, namely the effects of *increasing the* *lower bound* with regard to a fixed upper bound and *increasing the upper bound* with regard to fixed window sizes.

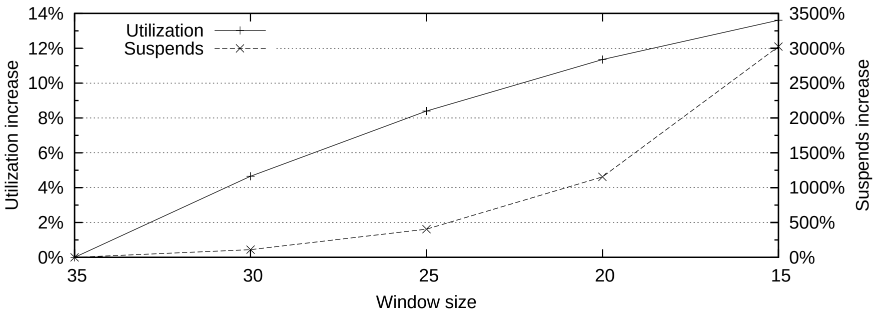

Fig. 5. Increase in utilization and suspensions when using 2 overbooking slots and an upper bound of 95. The lower bound is increased to decrease the overbooking window.

Increasing lower bounds: The first trend is the effect obtained by increasing the lower bound and keeping all other parameters constant. This results in utilization gains that slowly decrease per step. At the same time we find there is an exponential increase in negative effects. This is illustrated in figure 5, the lower bound is increased in steps of 5 from 60 to 80 creating corresponding overbooking windows [35:15]. The results show that although increasing the lower bound will give better utilization gains, these come at an increasingly higher cost. Figures 6 and 7 further show that this effect is present in all window, upper bound combinations.

Increasing upper bounds: Increasing the upper bound under a fixed window size results in a linear increase in utilization (see Figure 6) while suspensions (and degradation points) remain at roughly the same magnitude (see Figure 7). From these results we find that choosing a higher upper bound will increase utilization while having a small impact on the negative effects of overbooking.

In summary, we find that selecting a correct amount of *overbooking slots* is an important part of achieving optimal results. There is a tipping point where extra slots will only add negative effects without additional gain in utilization. We also find that increasing the *lower bound* has diminishing effects on utilization gains while negative effects increase exponentially. On the other hand, increasing the *upper bound* in our current simulator does not add negative effects while utilization displays a steady increase. This leads us to believe that a correct upper bound will most likely depend on limiting factors not yet explored in this research[3]. We can however conclude that the upper bound should

[3] In multi core systems with more VMs than cores, performance degradation will occur somewhere before total utilization hits 100%.

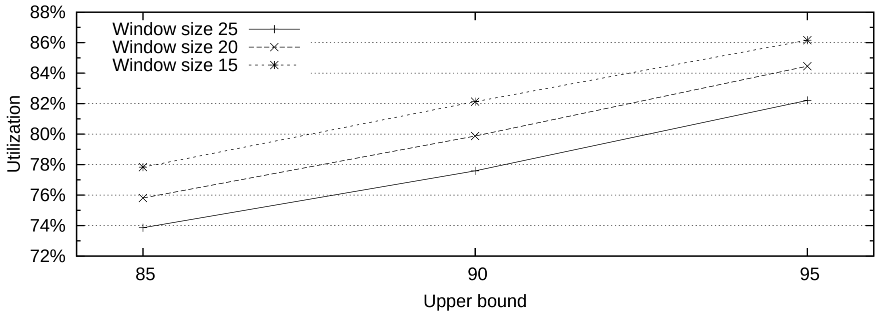

Fig. 6. Utilization with two overbooking slots and varying upper bounds.

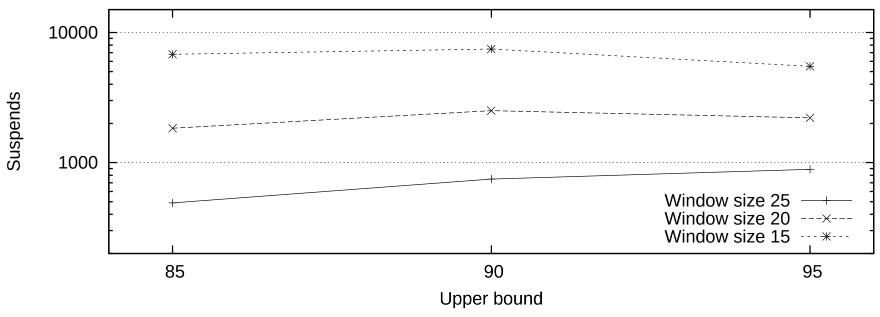

Fig. 7. Suspensions with two overbooking slots and varying upper bounds and windows.

be placed as high as possible. Depending on the amount of negative effects an administrator is prepared to allow, an optimal set of bounds can be chosen to maximize utilization.

5 Related Work

To deal with underutilization in batch queuing systems, backfilling techniques such as EASY [10] are often used. Jobs can jump ahead in the queue if they do not delay the start time of the first job in the queue. Conservative backfilling approaches [11] require that upon a backfill operation, no job in the queue is delayed in order to maintain fairness. A problem with these approaches is their reliance on user estimates of job runtimes which are often incorrect [12]. Several techniques have been proposed to model this runtime in order to tackle this problem [13–15].

Aside from backfilling, overbooking of resources is another technique to deal with underutilization. The scheduler deliberately overbooks resources in order to deal with jobs that do not use their allocated resource share fully. Sulistio et. al [16] developed a resource overbooking scheme for a setting in which resource reservations are made on a grid infrastructure. Whereas our work hinges on the

Bounds	Slots	Utilization	Hours	Suspends	> 100%	> 95%
85 - 60	1	73.7	3297.79	405	0	1
85 - 65	1	75.19	4434.55	1125	0	3
85 - 70	1	76.36	5338.98	3443	0	17
90 - 60	1	74.57	3966.83	191	0	10
90 - 65	1	76.3	5290.09	406	0	25
90 - 70	1	77.42	6150.38	1033	0	62
90 - 75	1	78.28	6806.58	2508	6	195
95 - 60	1	75.25	4491.03	144	3	144
95 - 65	1	77.29	6043.97	234	7	234
95 - 70	1	78.45	6936.92	383	13	384
95 - 75	1	79.17	7478.31	660	22	660
95 - 80	1	79.66	7864.45	1305	64	1305

Table 1. The results using different bound combinations and one overbooking slot.

Bounds	Slots	Utilization	Hours	Suspends	> 100%	> 95%
85 - 60	2	73.86	3418.20	490	0	1
85 - 65	2	75.81	4909.43	1839	0	9
85 - 70	2	77.83	6457.53	6797	3	81
90 - 60	2	75.07	4341.34	243	0	15
90 - 65	2	77.59	6277.52	746	1	49
90 - 70	2	79.87	8020.48	2507	7	196
90 - 75	2	82.13	9760.40	7461	49	868
95 - 60	2	75.84	4934.74	176	7	177
95 - 65	2	79.37	7639.71	370	18	371
95 - 70	2	82.21	9813.01	888	43	888
95 - 75	2	84.45	11528.12	2207	135	2208
95 - 80	2	86.16	12845.09	5498	577	5498

Table 2. The results using different bound combinations and two overbooking slots.

Bounds	Slots	Utilization	Hours	Suspends	> 100%	> 95%
85 - 60	3	73.88	3429.17	476.0	0.0	1.0
85 - 65	3	75.81	4915.34	1866.0	0.0	10.0
85 - 70	3	77.84	6468.15	6871.0	6.0	94.0
90 - 60	3	75.0	4289.79	242.0	0.0	16.0
90 - 65	3	77.57	6266.12	757.0	1.0	51.0
90 - 70	3	79.89	8047.17	2619.0	13.0	211.0
90 - 75	3	82.35	9921.47	8502.0	107.0	1190.0
95 - 60	3	75.88	4964.69	172.0	8.0	173.0
95 - 65	3	79.38	7643.12	376.0	18.0	376.0
95 - 70	3	82.36	9926.95	966.0	53.0	966.0
95 - 75	3	84.92	11885.76	2898.0	246.0	2899.0
95 - 80	3	87.33	13733.74	8838.0	1376.0	8838.0

Table 3. The results using different bound combinations and three overbooking slots.

exploitation of the volatility of VM workloads, their model attempts to deal with the binary case wherein reservations are not used at all or are canceled. They use a richer model for the cost of overbooking by introducing a penalty model that is linked to a renumeration, whereas we only consider the number of performance degradation points the schedule generates. In future work, we are interested in including such an application-specific penalty model to diversify the loss of value an application faces if it is subject to a degradation in performance.

An approach to overbooking non-preemptive workloads in a non-virtualized setting was proposed by Urgaonkar et al. [17]. They demonstrated that controlled overbooking can dramatically increase utilization on shared platforms. Resource requirements are based on detailed application profiling combined with guarantees requested by application providers. The profiling process requires all applications to run on a set of isolated nodes while being subjected to a realistic workload, this workload generates a set of parameters that must be representative for the entire application lifetime. Instead of pro-actively managing overbooking, application placement is based on a set of constraints and a probability with which these constraints may be violated.

Perhaps somewhat surprisingly, workload traces from the LCG-2 infrastructure, which supports the data processing of CERN's Large Hadron Collider, have shown that as much as 70% of the jobs run by a Tier-2 Resource center in Russia use less than 14% of CPU-time during their lifetime [18]. On the other hand, 98% of the jobs use less than 512MB of RAM. Cherkasova et al. thus investigate the potential of running the batch workloads in VMs and overbooking grid resources to increase utilization. The authors conclude that the use of virtualization and multiplexing multiple VMs on a single CPU core allows for a 50% reduction in the required infrastructure while rejecting less than 1% of the jobs due to resource shortage.

Birkenheuer et al. [19] tackle underutilization for queue-based job scheduling by modeling the probability that a backfill operation in the job queue delays the execution of the next job due to bad user runtime estimations or resource failure. A threshold is defined on this probability to decide whether a job can be used for backfilling. Birkenheuer et al. report on a 20% increase in utilization on a schedule for a workload trace of a 400 processor cluster. Their work is however not adopted to the specifics of virtual machine scheduling and only considers a single-processor case.

At the level of the VMM, priorities and weights can also be assigned to VMs such that high priority workloads maintain their resource share in the presence of low priority loads [20]. The VMM scheduler operates in time quanta that are in the order of tens of milliseconds to ensure the system allocates resources under the configured allocation constraints. Our approach differs from this in that we suspend virtual machines so that their memory pages can be reclaimed by other VMs. Although memory overcommitment is possible in popular VMMs such as Xen, HyperV and VMware, this can result in noticeable performance degradation if the VMs actually require the overcommitted memory [21, 22].

6 Future Work

Our first direction of future work will be to further evaluate the effectiveness of the presented scheduling approach. To obtain a complete view a larger amount of slot, bound and scenario combinations must be evaluated. Likewise, we wish to extend the set of workloads that are analyzed and, if possible, make use of trace data from real workloads. Our second goal is to classify VM workloads into predefined classes so that an optimal scheduling configuration can be chosen automatically. Thirdly, we want to improve the scheduling algorithm itself. In this regard we plan to explore the potential of workload modeling and prediction techniques to attain a more intelligent mapping between a low-QoS workload and the cluster node it is placed on. Finally, we plan to add aspects such as memory and network usage to the model in order to increase the accuracy of our results and to allow for the development of more complete scheduling. Using this more accurate model, we will compare our simulation results to those from OpenNebula experiments conducted with a real backend. In this manner, inconsistencies in the model and its assumptions can be rectified providing a realistic basis for further research.

7 Conclusion

We have introduced a scheduling algorithm which multiplexes low- and high-QoS workloads on a virtualized cluster infrastructure in order to increase the infrastructure's utilization through overbooking. By monitoring the difference between formal and actual requirements of high-QoS workloads in terms of CPU load, an opportunity to add low-QoS workloads to a cluster node is detected. We introduce a limited set of parameters in our scheduling policy so that a flexible tradeoff can be made between maximization of infrastructure utilization and workload interference. The results obtained from initial testing show that depending on the requirements, optimal parameters can be selected that significantly increase utilization while causing limited interference with high-QoS workloads. We identified general trends in the system's performance through parameter tuning and identified a number of guidelines to determine an optimal parameter setting.

References

1. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. SIGOPS Oper. Syst. Rev. **37**(5) (2003) 164–177
2. Weiss, A.: Computing in the clouds. netWorker **11**(4) (2007) 16–25
3. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R.H., Konwinski, A., Lee, G., Patterson, D.A., Rabkin, A., Stoica, I., Zaharia, M.: Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley (2009)
4. Amazon: Elastic compute cloud. `http://aws.amazon.com/ec2` (2008) [Accessed 22-12-08].

5. LLC, A.W.S.: Amazon ec2 spot instances (2009) [Accessed 23 December 2009].
6. Sotomayor, B., Montero, R.S., Llorente, I.M., Foster, I.: Virtual infrastructure management in private and hybrid clouds. IEEE Internet Computing **13**(5) (2009) 14–22
7. Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., Zagorodnov, D.: The Eucalyptus open-source cloud-computing system. In: 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID). Volume 0., Washington, DC, USA, IEEE (2009) 124–131
8. Sotomayor, B., Keahey, K., Foster, I.: Combining batch execution and leasing using virtual machines. In: HPDC '08: Proceedings of the 17th international symposium on High performance distributed computing. (2008) 87–96
9. Abrahao, B., Zhang, A.: Characterizing application workloads on cpu utilization for utility computing. Technical Report HPL-2004-157, Hewllet-Packard Labs (2004)
10. Feitelson, D.G., Jette, M.A.: Improved utilization and responsiveness with gang scheduling. In: Scheduling Strategies for Parallel Processing, volume 1291 of Lecture Notes in Computer Science, Springer-Verlag (1997) 238–261
11. Feitelson, D.G., Weil, A.: Utilization and predictability in scheduling the ibm sp2 with backfilling. In: IPPS '98: Proceedings of the 12th. International Parallel Processing Symposium on International Parallel Processing Symposium, Washington, DC, USA, IEEE Computer Society (1998) 542
12. Mualem, A.W., Feitelson, D.G.: Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling. IEEE Transactions on Parallel and Distributed Systems **12** (2001) 529–543
13. Tsafrir, D., Etsion, Y., Feitelson, D.G.: Backfilling using system-generated predictions rather than user runtime estimates. IEEE Trans. Parallel Distrib. Syst. **18**(6) (2007) 789–803
14. Verboven, S., Hellinckx, P., Arickx, F., Broeckhove, J.: Runtime prediction based grid scheduling of parameter sweep jobs. In: APSCC '08: Proceedings of the 2008 IEEE Asia-Pacific Services Computing Conference, Washington, DC, USA, IEEE Computer Society (2008) 33–38
15. Smith, W., Foster, I.: Using run-time predictions to estimate queue wait times and improve scheduler performance. In: Scheduling Strategies for Parallel Processing, Springer-Verlag (1999) 202–219
16. Sulistio, A., Kim, K.H., Buyya, R.: Managing cancellations and no-shows of reservations with overbooking to increase resource revenue. In: CCGRID '08: Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid, Washington, DC, USA, IEEE Computer Society (2008) 267–276
17. Urgaonkar, B., Urgaonkar, B., Shenoy, P., Shenoy, P., Roscoe, T., Roscoe, T.: Resource overbooking and application profiling in shared hosting platforms. (2002) 239–254
18. Cherkasova, L., Gupta, D., Ryabinkin, E., Kurakin, R., Dobretsov, V., Vahdat, A.: Optimizing grid site manager performance with virtual machines. In: WORLDS'06: Proceedings of the 3rd conference on USENIX Workshop on Real, Large Distributed Systems, Berkeley, CA, USA, USENIX Association (2006) 5–5
19. Birkenheuer, G., Brinkmann, A., Karl, H.: The gain of overbooking. In: JSSPP. (2009) 80–100
20. Cherkasova, L., Gupta, D., Vahdat, A.: Comparison of the three cpu schedulers in Xen. SIGMETRICS Perform. Eval. Rev. **35**(2) (2007) 42–51
21. VMware: Performance best practices for vmware vsphere 4.0 (2009)
22. Microsoft: Virtualization reality: Why microsoft virtualization solutions deliver value when compared to vmware (2009)