Improving Cloud Infrastructure Utilization through Overbooking

Luis Tomás
Dept. of Computing Science, Umeå University
SE-901 87, Umeå, Sweden
luis@cs.umu.se

Johan Tordsson
Dept. of Computing Science, Umeå University
SE-901 87, Umeå, Sweden
tordsson@cs.umu.se

ABSTRACT

Despite the potential given by the combination of multi-tenancy and virtualization, resource utilization in today's data centers is still low. We identify three key characteristics of cloud services and infrastructure as-a-service management practices: burstiness in service workloads, fluctuations in virtual machine resource usage over time, and virtual machines being limited to pre-defined sizes only. Based on these characteristics, we propose scheduling and admission control algorithms that incorporate resource overbooking to improve utilization. A combination of modeling, monitoring, and prediction techniques is used to avoid overpassing the total infrastructure capacity. A performance evaluation using a mixture of workload traces demonstrates the potential for significant improvements in resource utilization while still avoiding overpassing the total capacity.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Cloud Computing;
D.2.11 [**Software Architecture**]: Domain-specific architectures

General Terms

Algorithms, Management, Performance

Keywords

Admission Control; Burstiness; Cloud Computing; Overbooking; Prediction; Profiling; Resource Utilization;

1. INTRODUCTION

Efficient resource utilization has been a goal since long in data centers, motivated by hardware and operational costs [7] and lately also by power consumption and environmental concerns [15]. Virtualization technologies give data center providers increased flexibility in management of IT infrastructure. Similarly, the very large scale and multi-tenant nature of cloud infrastructures brings great potential for efficient multi-plexing and very high resource utilization. However, the cloud paradigm also introduce new obstacles for efficient resource management, as illustrated by recent data center workload studies that suggest low utilizations in overall. An analysis of Google traces [21] concludes that only 53% of the available memory is used whereas CPU utilization is as low as 40% on average. One reason for this is that users tend to overprovision their needs for the sake of safest execution. In a similar study where 5000 servers were observed for 6 months, Barroso et al. [2] report 10-50% as common levels for CPU resource utilization.

The most notably characteristic of cloud infrastructures is the elastic application nature. As these cloud applications make use of more or less resources over time, efficient resource allocation becomes significantly more difficult. In the case of horizontal elasticity, Virtual Machines (VMs) are dynamically (de)allocated based on changes in service workload [1]. This pattern in commonly used for interactive multi-tier services. From a cloud provider perspective, deciding how much resources to dedicate to an elastic service constitutes a challenging problem with significant impact on resource utilization on one hand, and potential Service Level Agreements (SLAs) violations on the other. For the rest of this paper, we view SLAs as defined by availability service (e.g., 97% of the time) and a penalty in case this is not fulfilled.

In cloud environments, VMs are usually configured during creation with a specific amount of resources, such as CPU, disk, and memory [18]. Over-provisioning those VMs leads to waste of resources, from the cloud provider perspective, and higher costs, from the user perspective. Conversely, under-provisioning may result in performance degradation. When this is done by the users, there is no problem from the provider perspective. Nevertheless, if the provider is under-provisioning to increase the infrastructure utilization, then the resulting performance degradation may result in loosing customers. Furthermore, providers usually only offer predefined VM sizes (e.g., S, M, L, XL) with a fixed amount of CPU, memory, disk, etc., which the user is not allowed to customize. Thus, in order to meet the requirements of the applications, users have to select the VM size that provides enough of the most critical resource type (such as CPU), while typically over-provisioning other resources, e.g., disk, memory, and network bandwidth [10].

Another aspect of elasticity is that cloud applications (encapsulated into VMs) do not use the same amount of hardware resources (CPU, memory, I/O, etc.) all the time, commonly referred to as vertical elasticity. This can either be related to different behavior due to internal phases of application operation, or be caused by variations in incoming workload. Users tend to provision the worst-case capacity as the upper bound (or even more for leaving a margin), even though that amount of CPU, memory, or I/O is only used for a small period of time. A related study of parallel computing workloads suggest that more than half of all jobs use less than 20% of requested capacity [6].

In summary, the three main characteristics of cloud infrastructure that tend to reduce resource utilization are:

1. Horizontal application elasticity - changing the number of VMs allocated to a given service over time.

2. Pre-defined sizes of VMs - forcing application developers to waste resources.

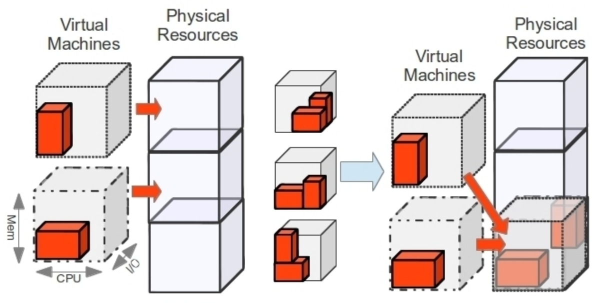
3. Vertical application elasticity - changes in the actual resources (CPU, bandwidth, etc.) usage by each VM over time.

1.1 An Overbooking Approach

The above listed cloud infrastructure characteristics combined with users' tendency to overprovision resources to be on the safe side result in significant waste of resources in a cloud infrastructure. *Overbooking* describes resource management in any manner where the total available capacity is less than the theoretical maximal requested capacity. This is a well-known technique to manage scarce and valuable resources. Overbooking has been applied in various fields as diverse as airline yield management [22], network bandwidth allocation [11], and batch scheduling for parallel computers [4]. To address the cloud infrastructure utilization problem, we propose a two-pronged overbooking strategy. To cope with horizontal elasticity, we use *admission control* strategies where the maximum number of VMs (aggregated over all services) accepted into a data center is larger than the total infrastructure capacity. To handle the effects of vertical elasticity and pre-defined VM sizes, we propose *scheduling* techniques that overbook each physical server of the infrastructure. Long-term capacity planning (admission control) and short-term resource scheduling must be studied in combination as the precision of admission control greatly impacts scheduling. Having too few VMs in the data center makes scheduling trivial and in-efficient, but with too many VMs it is impossible to find, with or without overbooking, a suitable allocation of these.

A conceptual overview of cloud overbooking is shown in Figure 1. It can be seen that if two applications ask for some capacity (gray boxes) but they actually use less than the agreed amount (red boxes). Now, if and only if the total real capacity being used for both applications is less than the real capacity available at one node, the VMs running those applications may be co-allocated at the same physical resource without (noticeable) degradation in their performance. The three dimensions (CPU, mem, I/O) must be taken into account.

Overbooking techniques always exposes the cloud provider to a risk of resource congestion and consequently of SLA violations. Thus, we need to establish a trade off between improving the resource usage and exposing the infrastructure provider and customers to the risk of resource congestion and performance degradation. Consequently, one



(a) Without overbooking. (b) With overbooking.

Figure 1: Conceptual illustration of overbooking advantage showing how two small enough VMs can be collocated without disturbing each other.

of the main challenging issues when dealing with resource overbooking is how to decide the appropriate amount of excess capacity to allocate to minimize the risk of SLA violations [23]. When providing IaaS, those SLA violations may become an issue due to total compensation cost, legal and regulatory issues to address in some cases and, most importantly, market acceptance when provided quality is not as expected. The real challenge addressed in this paper is hence how to estimate the total capacity needs and ensure that applications performance is not degraded due to overbooking, while at the same time minimize resource wastage. It must therefore be noted that the main aim of this work is not to change the SLAs to enable higher utilization, but to increase the number of SLAs that the infrastructure accepts without risking the ones already agreed and in a transparent way to the users. Hence, we try to keep the capacity used below the physical capacity limits to avoid performance degradation that would lead to not meeting SLAs.

Resource characteristics and application behavior both have to be taken into account when deciding which resource types to overbook. Most applications run, albeit slower, if allocated too little CPU, whereas provisioning too little memory commonly makes applications crash. Regarding application characteristics, e.g., CPU capability cannot be treated in the same way for applications that require high throughput as for ones that require low response time. This is similar to other capabilities, such as network usage, where bandwidth and latency are the equivalent parameters. To exemplify, for MapReduce services, high bandwidth and throughput may be preferred (reducing the shuffling and execution phase, respectively), whereas for interactive services, low latency and response times are the most critical aspects.

The rest of the paper is organized as follows. The proposed overbooking system, its main components, and algorithms are detailed at Section 2. Section 3 describes the experimental evaluation where the efficiency of the proposed system is evaluated by a set of simulations. Finally, Section 4 discuss related work, followed by the conclusions and future directions at Section 5.

2. SYSTEM PROPOSAL

In order to study the afore-mentioned overbooking capa-

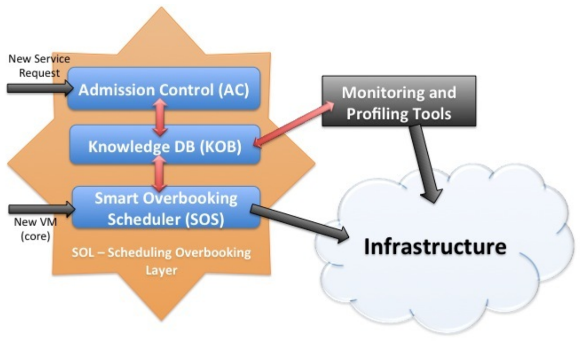

New Service Request
Admission Control (AC)
Knowledge DB (KOB)
Smart Overbooking Scheduler (SOS)
New VM (core)
SOL – Scheduling Overbooking Layer
Monitoring and Profiling Tools
Infrastructure

Figure 2: System Proposal Snapshot with the three main components (AC, KOB and SOS) and their relationship with the cloud infrastructure, and monitoring and profiling tools. AC decides whether allocate the new request; SOS which is the best placement; and both of them use the information collected by KOB.

bilities, we have implemented a cloud capacity management framework. Our system handles the three main tasks of self-management: data collection, forecasting, and analysis. The proposed system is depicted in Figure 2. Data collection is handled through different monitoring tools, which are in charge of monitoring the status of either physical and virtual machines and of profiling the running workloads. This data collection process is carried out by several (interchangeable/customizable) plug-ins managed by the ***Knowledge DB (KOB)*** module.

Based on the collected monitoring data, we forecast expected future resource (CPU, memory and I/O) of the system. Then, the *analysis* of all available and predicted data is used in long-term capacity decisions, i.e., whether to accept incoming applications or not (***Admission Control (AC)*** module). The collected monitoring data and the subsequent forecasts and analysis are also used for shorter-term capacity management. This scheduling is used to decide which node(s) and/or core(s) are the most suitable ones for allocating the VMs belonging to the newly admitted application (***Smart Overbooking Scheduler (SOS)*** module).

Forecasting techniques are a must to estimate the future capacity requirements of provisioned VMs. Significant trends must be detected but at the same time, we must avoid taking management decisions based on temporary load spikes only. To this end, time-series forecasting techniques are used (triple exponential smoothing function [24]), and combined with statistical modeling to infer future usage and workload patterns, respectively.

Detailed information about each one of the three main modules is described at next subsections. Table 1 gives an overview of the used notation.

2.1 Admission Control

The Admission Control module decides whether a new service/application deployment request should be accepted or not. Thus, by using the information provided by the knowledge DB module, AC has to evaluate the impact that accepting this new request will have at the short and long term system behavior, i.e., weighting improved utilization against the the risk of SLA violations.

Table 1: Notation.

App	Incoming Application
$AppProfile$	App Profile
$VMType$	VM Type required by App
N	Set of nodes $\{n_i$ / i in [1..m] $\}$
TC	Total Capacity
TNC	Total Node Capacity
C_{used}	Capacity already booked
RC_{used}	Real Capacity in use
$R_{n_i}C_{used}$	Real node n_i Capacity in use
Min	Minimum # of VMs needed by the App
Max	Maximum # of VMs needed by the App
Avg	Average # of VMs needed by the App
OBF	Overbooking Factor
OBF_{n_i}	Overbooking Factor of node n_i
$ObjFunction$ $(Min, Max, Avrg)$	Returns # of VMs to book depending on the selected objective, Min, Max or Avg
$Prediction(x)$	Predicts future values of x
$Area(r, q)$	Area of line r over line q
$GetSlopeValues(y)$	Obtains final and max slope for time series y

Algorithm 1 Non Overbooking Admission Control

```
1: $Requested_{VMs} = ObjFunction(Min, Max, Avrg)$
2: if $C_{used} + Requested_{VMs} <= TC$ then
3:     $C_{used}$ += $Requested_{VMs}$
4:     Accept $App$
5: else
6:     Reject $App$
7: end if
```

Algorithm 2 Overbooking Admission Control

```
1: if $Prediction(RC_{used}) + AppProfile <= TC$ and
   $OBF > OBF_{threshold}$ then
2:     Accept $App$
3: else
4:     Reject $App$
5: end if
```

In this work we want to evaluate the improvement obtained by the overbooking techniques as such. Thus, we use only two simple admission control policies: with and without applying overbooking. Basically, for the non overbooking case, AC is based on accepting the applications (of n VMs of type t) by taking into account the number of cores requested: *maximum*; *minimum*; and *average*. This process is detailed at Algorithm 1.

For the overbooking policy, there are two different implementations. The first one where no admission control at all is performed, then all the requests are accepted. In the second one, the decisions are taken based on several parameters to avoid overpassing the real capacity. This algorithm (Algorithm 2) takes into account current and predicted status of the system (real usage, not requested resources), the workload profiles and the overbooking already achieved (for more details about how to calculate Overbooking Factors, see Section 2.3), but without analyzing the long term impact. This means that only the current status is considered when taking the decision, hence not considering the impact of having to deploy more VMs for the already accepted applications in the future.

2.2 Monitoring and Profiling Tools

Measuring and profiling different applications behavior, taking into account their different dimensions (CPU, memory and I/O) is a must for accurate analysis and decision making in admission control and scheduling. In this work, due to the lack of real available workloads and to be able to have reproducible experiments, workload profiling has been made offline and then resource executions emulated into simulated resources. However, through a plug-in architecture, the monitoring framework is easily integrated with monitoring frameworks for VMs and physical nodes, e.g., the Libvirt library [16] or Nagios [19], that allow us to measure the virtual and physical machines status. The same type of tools can be used for application performance profiling but more appropriately, more fine-grained monitoring should be used.

When overbooking, the different applications profiles must be taken into account [26]. For instance, the behavior of a web service application differs from that of a gaming server. The latter one has latency requirements making it less tolerant to violations of performance guarantees than the web service. In addition, some applications are more suitable of being collocated with other VMs. As highlighted in [9], workloads that present a bursty (peaky) behavior are really prone to be overbooked since they only occasionally use all the system resources that they are entitled to. Peaks and lows in one workload do not need to coincide with the ones at others workloads, thus they would be a good match for being co-allocated. Moreover, as VMs are multidimensional (CPU, memory, I/O), we cannot only take advantage of allocating bursty and non bursty applications together, but also of co-allocating CPU-bound and network-bound or memory-bound applications [26].

2.3 Smart Overbooking Scheduler

Once the AC has decided that an application has to be deployed, the scheduler is in charge of deciding which is the most suitable node and core(s) for each VM. As physical servers have limited CPU, memory, and I/O capabilities, these have to be carefully considered when performing the overbooking to try to avoid placements that may lead to low performance. As highlighted in [26], making the overbooking by only taking into account average resource requirements or only taking into account one dimension can result in significantly reduced performance. This is why we use a set of different parameters when deciding where to schedule VMs. The scheduling process is detailed at Algorithm 3.

This worst-fit style algorithm predicts first what is the future expected usage of the physical resources – to just take into account the real usage, not the requested one. To obtain those predictions a triple exponential smoothing function is used since it has been shown to provide accurate enough results [24]. This information is used together with the application profile to estimate if accepting the new incoming request would overpass the total real capacity in any of the dimensions (line 4). Figure 3 illustrates this graphically, where the gray area shows the estimated time during which the real available capacity is estimated to be less than the required one, increasing the possibilities of SLA violations, delays, failures, etc.

If that area is small enough (Line 5), the framework then takes into consideration how overbooked the selected node already is (Line 6) and what is the trend of that overbooking (Line 7). To measure the overbooking already done we define an overbooking factor as:

$$OBF_X = \frac{(UsageRequested_X - RealUsage_X)}{min(UsageRequested_X, RealCapacity_X)}, \quad (1)$$

Algorithm 3 Worst-Fit Overbooking Scheduling

```
1: Allocated = false
2: NS = Sort Nodes $N$ by $OBF$
3: for each $n_i \in NS$ do
4:   $AccumulatedUsage = Prediction(R_{n_i} C_{used}) + AppProfile$
5:   if $Area(AccumulatedUsage, TNC) < Area_{threshold}$ then
6:     if $OBF_{n_i} > OBF_{threshold}$ then
7:       $finalSlope, maxSlope = GetSlopeValues(Prediction(OBF_{n_i}))$
8:       if $|finalSlope| < Slope_{threshold}$ and $|maxSlope| < Slope_{threshold}$ then
9:         Allocated = True
10:        AllocateVM at Node $n_i$
11:      end if
12:    end if
13:  end if
14: end for
15: if $Allocated == false$ then
16:   AllocateVM at Node with highest OBF
17: end if
```

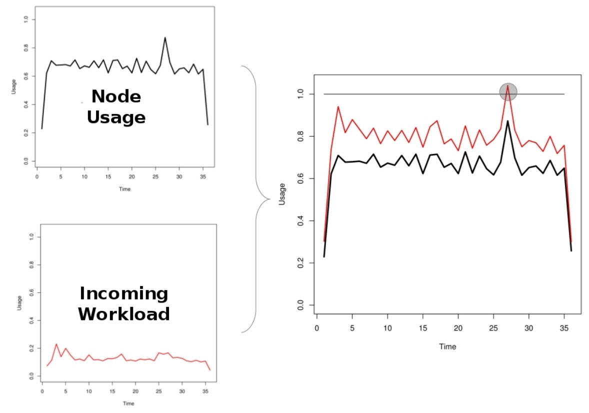


Figure 3: Overbooking decision. The gray area means the predicted risks that will be taken if the application is allocated in that node.

where X can be CPU, Mem, or I/O, depending on the capacity we are measuring. This overbooking factor is calculated for each node and core in the system (data center) and for each dimension. The range of OBF_X is (0,1) since $RealUsage$ cannot be greater than the requested one as it is encapsulated within the requested VM and $RealUsage$ cannot be negative. Thus, OBF values represent how overbookable the resources are: the greater the value the higher the potential for overbooking.

After calculating overbooking factors for all hosts, we use a worst-fit (WF) technique to select the least overbooked resource(s) (Line 2). We also implemented best-fit and first-fit methods, but found these inferior. Due to lack of space this comparison is omitted and worst-fit is used. Urgaonkar et al. draw similar conclusions regarding worst-fit in their study of overbooking in shared hosting platforms [26]. To be able to take all resource dimensions into account in a single comparison in the worst-fit algorithm, the overbooking value for each resource (nodes and cores) is calculated by multiplying the OBF obtained for each dimension:

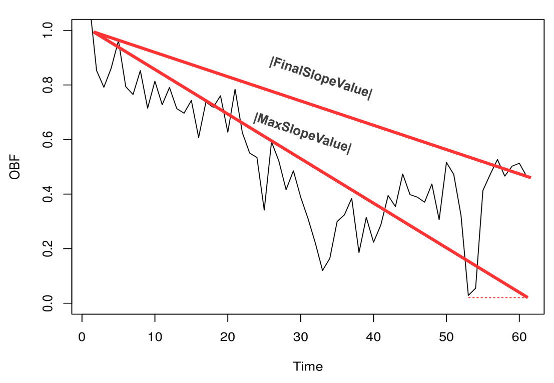


Figure 4: Slope-Aware algorithm for predicting the OBF for the next hour in the target node. It calculates the trend by taking into account the final expected OBF and the worst case during that period.

$$OBF = OBF_{CPU} * OBF_{Mem} * OBF_{I/O}. \qquad (2)$$

These values are also used to finally decide whether the overbooking action is going to be taken or not. To decide that, OBF values are predicted for the near future (Line 7). Then, it is checked that the OBF values are bigger than a certain threshold (customizable parameter depending on how risky we want to be when performing the overbooking actions) and that the tendency (depicted by $FinalSlopeValue$ and $MaxSlopeValue$ in Figure 4) is also below a certain absolute value (Line 8). Figure 4 shows a graphical description of this slope-aware algorithm. In that case, the resource is suitable for being overbooked. Otherwise, the next node has to be checked and if there is no node left, the node with highest total OBF is chosen for allocating the request. Notably, as admission control has accepted the new application, all VMs must be scheduled by SOS, even though this could result in too aggressive overbooking of certain hosts.

3. EXPERIMENTS

In this section we first present the emulated testbed and the workload generated to measure the performance of our proposed framework. After that, the actual performance evaluation is depicted. For simplicity, here we henceforth mostly focus on illustrating and analyzing the results for one capacity dimension (CPU – higher utilization than the others), although also memory and I/O have been taken into account in all experiments.

3.1 Simulated Infrastructure

As explained in Section 2.2, we have implement our proposal in a framework that is able to communicate with and monitoring real infrastructures, data center resources, virtual machines, etc., as well as working over a simulated one. This way, and due to unavailability of traces from actual executions or even characterization of the workloads that IaaS providers are running, we have emulated the behavior of workloads and carried out discrete event simulations of the

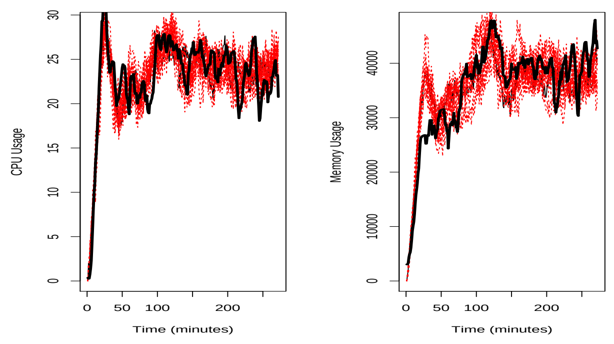


Figure 5: Simulator behavior for CPU (left) and memory (right). Simulated resources (red dashed lines) have a similar behavior as the real one (black line).

Table 2: Virtual Machines sizes.

	# CPUs	Memory (MB)	Bandwidth (Mbit/s)
S	1	2048	1000
M	2	4096	2000
L	4	8192	4000
XL	8	16384	8000

resources that execute them to study the demand imposed on our system: we have simulated the physical resources belonging to the data center and emulate several applications that to the best of our knowledge are representative within cloud environments and consequently relevant for real cloud providers (see Section 3.2).

The cloud infrastructure simulated for testing our algorithms consists of 16 Nodes where each one of them has 32 Cores. Those cores simulate the execution of the workloads modeled in next section by following the profiled usage of them. Here it must be noted that the bigger the infrastructure is, the better improvements we could achieve since the overbooking techniques take more advantage of the three properties (horizontal and vertical applications elasticity, and pre-defined VM sizes) discussed at Section 1.

On the other hand, the reason for selecting an infrastructure of X nodes of 32 cores is based on the fact that we have a cluster with those characteristics (32 AMD Opteron(TM) Processor 6272, 64 GB Memory) where we have confirmed that our simulator performs similar to the real server. This can be seen at Figure 5, where real server (black line) has similar trend than the simulated resources and workloads (red dashed lines).

We consider four different types of VMs, similar to Amazon model, with the characteristics detailed at Table 2. Even though those four different kind of VMs can be used by all applications, the results shown here are for fairness reasons using only VMs of S type. If VMs of bigger sizes are used, the improvements obtained thanks to overbooking actions would be bigger. For instance, if we use a bursty application that sometimes use 1 cores and others 8 and map that into a XL VM, then we can take advantage of the 7 unused cores (from time to time) for achieving greater consolidations.

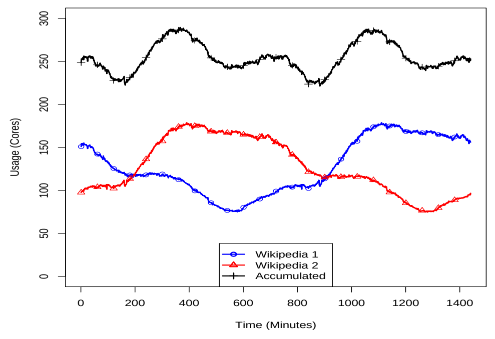


Figure 6: Background workload: Web servers showing resource requirements of two Wikipedia traces over time as well as the accumulated workload.

3.2 Workload Modeling

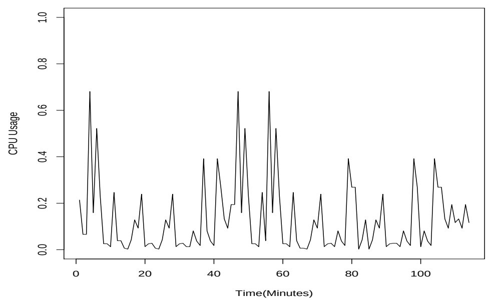
As highlighted before, there are no detailed enough workloads available from real cloud environments. For this reason we have emulated the following two class of workloads:

- *Background workload*: web server applications with varying number of user requests. This workload is interpolated from real available traces, in this case Wikipedia traces [20].
- *Dynamic workload*: applications profiled by using monitoring tools after running the real application and generating a workload through a poisson distribution. Two different type of applications are profiled, one with steady behavior and the other one with bursty usage.

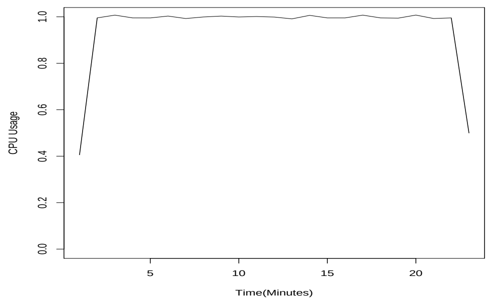
With those workloads we have emulated a scenario that could be close to reality to the best of our knowledge, composed of a mixture of background and dynamic workloads. The reason of choosing this mix as the workload to test our proposal is to have different kind of applications (with steady and bursty behavior), as well as having services that present (require) some elasticity in their usage – sometimes need a few resources and other times a large amount of them. This way we can exploit horizontal and vertical elasticity problems regarding resources utilization.

For the first category (a web server) we have used available information about Wikipedia traces [20]. Two different one-day periods were selected and extrapolated to our environments, i.e., changing the available information (number of requests per seconds) into resources usage (CPU, memory and I/O usage). The information about how many cores are required by this workload is depicted at Figure 6. As can be seen, for supporting that kind of requirements each web server must have between 80 and 180 cores serving requests depending on the current load. Therefore, we assume that the web workload owner wants to establish a contract (SLA) with the cloud provider specifying that an amount between 100 and 200 cores (VMs of S type in this case) have to be ready for attending the possible user requests – giving some extra margin for unpredictable situations.

Regarding the dynamic workload, we have modeled two different kind of jobs, with different length in time and behaviors. The first one (see Figure 7 (a)) represents an application with bursty behavior (such as a web server) whereas the second one (Figure 7 (b)) shows a totally different behavior – stable (steady), such as a CPU-bound map-reduce job. These two application profiles have been obtained by executing two different types of applications and using the LTTng2 [17] monitoring tools to measure their resource usage over time. This tool provides very detailed profiling of application behavior and supports integrated kernel and user-space tracing from a single user interface. These two different jobs are equally mixed and submitted into the system according to a Poisson distribution with 20 jobs per minute on average.



(a) Bursty application profile.



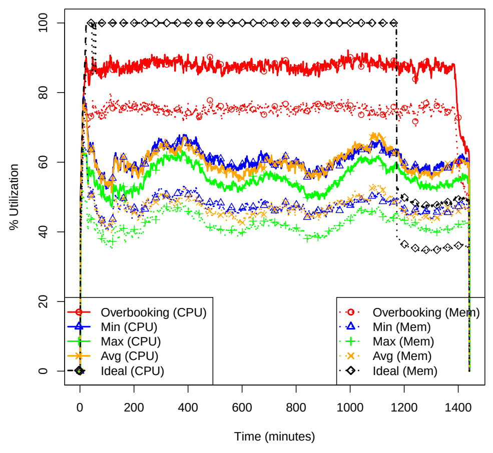
(b) Steady application profile.

Figure 7: Dynamic workload: Resource usage profile of a bursty (a) and a steady (b) application.

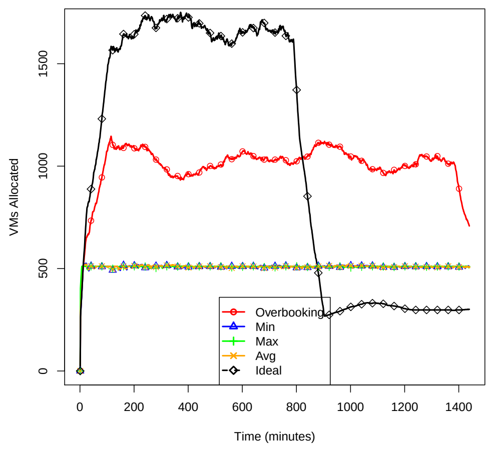
3.3 Performance Evaluation

In order to highlight the benefits of performing resource overbooking to increase resource utilization within a data center, the proposed overbooking algorithm is compared with different non overbooking techniques and with the ideal (unrealistic) case presented in Section 2.1. For the non overbooking techniques three approaches are compared: *Min*, *Max* and *Avg*. These means that when the Wikipedia web service application (which presents elasticity) is admitted into the system, it respectively deploys the minimum, maximum, and average number of VMs required by them, with the subsequent risks or resource wastage. The application can scale resource allocation up to Max and scale back down to the initial allocation. Notably, in the Max scenario there is no scale up or down at all.

Regarding the admission control techniques we have, on the one hand, the ideal method (labeled as *Ideal*) which accepts all the requests even though there is no space left into



(a) CPU and Memory Usage.



(b) Allocated VMs.

Figure 8: Comparison of resource usage (a) and number of VMs allocated (b) along time with and without overbooking.

the resources, also assuming a perfect sharing (without overhead) of the physical resources. On the other hand, we have our proposed technique (labeled as *Overbooking*) that takes decisions by analyzing the current impact of overbooking the resources and whose main objective is to remain close to this ideal scenario, but without overshooting the total real capacity to avoid SLA violations.

Figure 8 shows the results for those comparisons. Figure 8 (a) shows the CPU (solid lines) and memory (dotted lines) usages along time for each technique. The same information regarding the most congested dimension (i.e., CPU) is summarized at Table 3 where the total amount of dynamic workload jobs accepted is also highlighted. Figure 8 (b) shows the total amount of VMs allocated along time. Both of them depict the significant improvement obtained by using the overbooking technique (red lines), which increase resource usage between 44.7% and 56.6% regarding

Table 3: Real CPU Usage Summary (Figure 8 (a)).

	Overbooking	Min	Max	Avg
1st Qu.	443.4	299.8	271.1	294.4
Median	448.4	308.2	280.2	304.6
Mean	444.0	311.1	283.6	308.1
3rd Qu.	453.5	322.9	297.6	321.8
Max	472.1	405.5	331.4	399.5
Applications Accepted	15055	4542	2376	4303

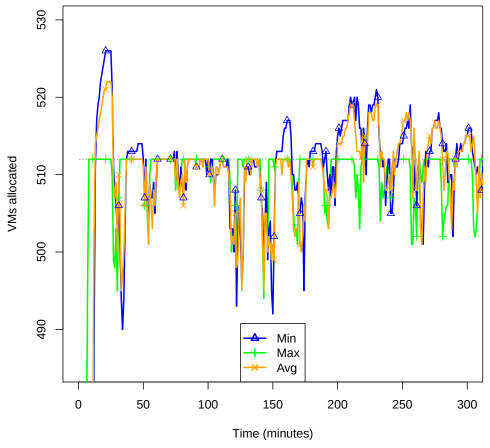


Figure 9: Allocation limit (no overbooking). Allocated VMs exceed total capacity due to elastic services.

CPU and between 55.8% and 76.1% for memory, leading to accepting between 3.3 and 6.3 times more dynamic workload applications in the same period of time. Even better results could have been obtained by not only using S type VMs.

As presented in Figure 9 (zoom of first 300 minutes of Figure 8 (b)), using Min or Avg techniques instead of Max when performing admission control of elastics web services could result in overshooting the capacity (without overbooking techniques) due to elastic scale up of the web server applications. This figure also shows how Max never passes that limit as the system books the whole amount of VMs needed (no elasticity). Conversely, the Min and Avg techniques overshoots every now and then by accepting too many jobs without forecasting the scale up needs of already allocated web servers.

Figure 8 also compares the implemented overbooking technique with the ideal upper limit (labeled as *ideal* in the figure, black lines). In that case the same amount of jobs are accepted (15055) without any admission control mechanism and without taking into account any overhead due to time resource sharing when real usage of accepted applications is greater than the real available capacity. This ideal case accepts that amount of jobs within the first 800 minutes and then, when all the accepted applications from the dynamic workload have been executed (around minute 1150), only provision the two Wikipedia servers – having therefore less load than any other technique from that point onward. Notably the proposed overbooking technique remains close to the ideal scenario. With a more advanced admission control algorithm, the framework could have performed more aggressive strategies, accepting a greater number of jobs leading to a better increment regarding data center utilization.

It must be noted that exceeding the capacity, regarding number of VMs already allocated, does not mean violating SLAs as long as the real capacity being used always remains below the total available physical capacity. Hence, there are problems for the "ideal" case since total capacity is less that required most of the times, although we are not considering them for that case to have a comparison with the (unrealistic) ideal case where all the resources are being used all the time. Therefore, our overbooking techniques would have no SLAs violations (at least none caused by overbooking) since total capacity being used in each dimension always remains below the total available capacity. As Figure 8 (a) depicts (red lines), CPU stays with a margin of approximately 10% below the real capacity and memory around 20%. The problems could appear when applying more aggressive algorithms that aims to increase the utilization even more (taking higher risks), leading to full utilization at some points. This could be a problem that may result in SLA violations regardless the duration of the overloaded period since some applications could not recover from that or simply not finish on time.

Furthermore, the application profile influences the improvement obtained since steady applications yield smaller gains when overbooking resources (if they are provisioned accurately enough). Additional gains are possible by collocating applications that present different bottlenecks, such as CPU intensive workloads with network demanding applications. We repeat the experiments in Figure 8 (a) to study these effects in more detail, we also present a comparison regarding the number of applications concurrently accepted when the proportion between bursty and steady applications is changed. Figure 10 presents the trend when the ratio between bursty and steady applications is 25%, 50% and 75%. This figure only shows 200 minutes since from that point usage remains stable for all the overbooking cases. Differences can be appreciated early in the experiment, but after some time they tend to achieve a similar and stable resource utilization. The initial difference among the overbooking ones stems from having more steady applications that are filling the resources (real capacity) faster than bursty ones. After that, thanks to the overbooking techniques, all three scenarios reach a similar usage regardless the distribution of bursty and steady applications. Consequently, when overbooking techniques are applied, the resource utilization remains stable independently of the workloads that are being submitted. By contrast, when no overbooking is used, the utilization varies along time and differs depending on the applications profile – the larger fraction of bursty applications the lower utilization.

Although the ratio of bursty and stable applications have little impact in the long term on the overall resource utilization, there are large differences regarding the number of VMs deployed, as depicted in Figure 11. Related information is again summarized at Table 4. The more bursty applications there are, the larger number of VMs can be concurrently deployed thanks to the overbooking techniques. This results in a gain of 8.6% from 25% to 50% of bursty applications and of 9.8% from 50% to 75% regarding the total number of accepted VMs. Regarding the number of concurrent VMs deployed at the same time, the number increases 36.9% on average from 25% to 50% ratio of bursty applications and around 32.3% from 50% to 75% burstiness ratios.

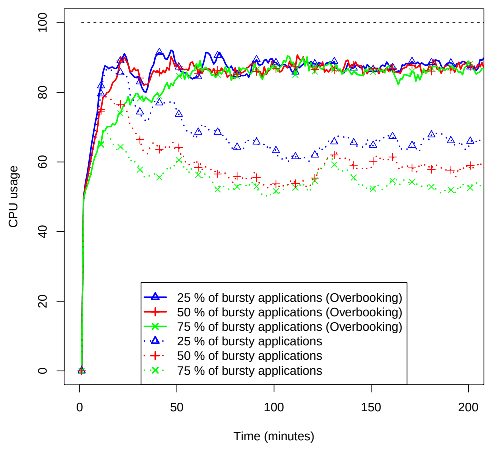


Figure 10: Utilization over time with different ratios of bursty applications. When overbooking techniques are used convergence rate is impacted for burstiness, but overall utilization is not.

Table 4: Allocated VMs Summary (Figure 11).

Percentage of bursty apps.	**25%**	**50%**	**75%**
1st Qu.	726	985	1281
Median	743	1028	1366
Mean	734.3	1005	1330
3rd Qu.	770	1055	1410
Max	815	1146	1598
Dynamic Workload Applications Accepted	13868	15055	16527

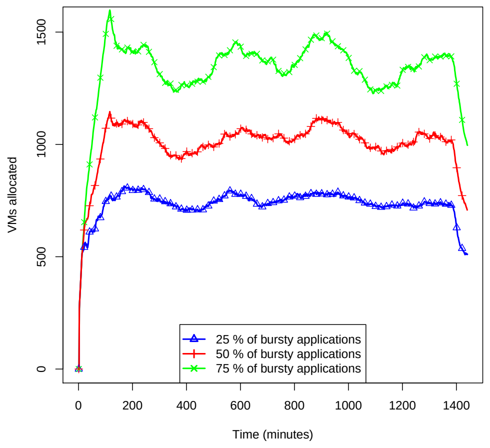


Figure 11: Number of VMs allocated over time using overbooking with different ratios of bursty and steady applications. Higher burstiness rate results in greater number of VMs allocated.

In summary, the improvement obtained thanks to the proposed overbooking techniques is significant even with simple admission control techniques. More advanced admission control would allow us to perform a more aggressive scheduling techniques with the subsequent increments in resource utilization.

4. RELATED WORK

Overbooking techniques as such have been applied in various fields as diverse as bandwidth allocation [11], airline yield management [22] and parallel computer scheduling [4].

Urgaonkar et al. propose techniques to overbook cluster resources in a controlled way that guarantees applications performance even despite overbooking [26]. However, they assume that users provide information regarding the degree of overbooking that their applications may tolerate as well as their time periods, which at times may be known by the users in a cluster environments but is not available for cloud infrastructures.

There are also more recent studies centered on cloud environments. In a paper by Ghosh et al. [9], the risks of overbooking resources in a cloud are analyzed and a threshold-based overbooking scheme is proposed. The trade-off between overbooking and performance degradation is closely related to SLA management. Similarly, Breitgand et al. [5] present an algorithmic framework that uses cloud effective demand to estimate the total physical capacity required for performing the overbooking. Their work propose to extent standard availability SLAs to also include probability of successfully launching additional VMs, but the model is based only on account CPU usage.

A key aspect of all overbooking systems is insight in future resource usage. The literature on resource behavior prediction within highly distributed systems such as Grids or clouds is very rich. A survey of several prediction techniques is presented in [8]. Examples of techniques include adaptive methods [13], state-space models [14], exponential smoothing [24], and use of control schemes for self-tuning for improved forecasts [25].

Another important aspect of overbooking is the suitability of co-allocating VMs into the same physical node(s). He et al. [12] present an algorithm for improving resource utilization for cloud providers based on a multivariate probabilistic model. The suitable physical hosts for VMs are selected based on the three dimensions (CPU, memory and I/O). VM (anti)affinity rules are used to avoid repeating poor performance in the future. A similar approach is taken by Meng et al. [18], who propose a joint VM provisioning approach that, based on estimates of the aggregate VM capacity requirements, allocates and consolidates VMs. Their work only takes into account CPU usage and perfect predictions about future workload behavior is assumed.

With a somewhat different aim than ours, Beloglazov et al. [3] propose a Markov chain model and a control algorithm for the problem of host overload detection as a part of dynamic VM consolidation. However, their work is based on detecting when hosts are overloaded and perform the needed migrations (to the less loaded resources), instead of focusing on scheduling and admission control. VM migration could serve as a valuable complement to these in case of prediction failures. Gmach et al. [10], address the fixed size VM problem of cloud infrastructure vendors, which they refer to as the "t-shirt" model. On the contrary, in some private cloud scenarios, the capacity of each VM is permitted to change dynamically using a time-sharing mechanism. Gmach et al. present a tool to help customers to decide which approach works most efficiently for their workloads. Their evaluation demonstrates that for a given set of workloads, the t-shirt model requires almost twice the number of physical servers as the time share model. However, resource overbooking is not used in their model.

5. CONCLUSIONS

VM consolidation overbooking is a promising solution to address the resource utilization problems that arise due to the elastic nature of cloud applications and current infrastructure management practices. However, it has to be carefully carried out to prevent performance degradation.

We propose a framework with admission control and scheduling mechanisms that are capable of resource overbooking without overpassing the total available data center capacity. Our experimental evaluation demonstrates that already with a simple admission control mechanism, resource utilization can be improved with more than 40%, and three times more applications can be admitted concurrently.

A general conclusion is that overbooking is more beneficial for the bursty applications. However, the system performance also depends on the accuracy of the underlying estimations regarding CPU, memory and I/O usage, as well as on how accurately the workloads have been profiled and how predictable their performance are. All these factors have a strong relationship with the potential resource utilization improvement achievable with overbooking.

As future work we plan to study affinity functions that aids the scheduling system in deciding which applications to collocate. This would allow the system to further increase the resource utilization and perform overbooking with less risk. Implementing a more advanced admission control functionality is another future direction. With more accurate forecasts of long-term resource usage of all applications, more aggressive overbooking strategies can be applied to further increase resource utilization. Another extension to the current work is to evaluate our system by executing real applications and infrastructures instead of using simulations based on traces. This would allow us to create improved overbooking models by quantifying the overhead associated with VM collocation, context switching, etc, as well as assess its impact on application performance and overall resource utilization or discovering unexpected effects that could lead to increment the number of SLA violations.

Acknowledgment

This work was supported in part by the Swedish Research Council under grant number 2012-5908. We thank Ahmed Ali-Eldin who provided valuable feedback on workload traces and elasticity. We also acknowledge the anonymous reviewers whose constructive comments improved the quality of this paper.

6. REFERENCES

[1] A. Ali-Eldin, J. Tordsson, and E. Elmroth. An adaptive hybrid elasticity controller for cloud infrastructures. In *Proc. of Network Operations and Management Symposium (NOMS)*, pages 204–212. IEEE, 2012.

[2] L. A. Barroso and U. Holzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, 2007.

[3] A. Beloglazov and R. Buyya. Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints. *IEEE Transactions on Parallel and Distributed Systems*, In press.

[4] G. Birkenheuer, A. Brinkmann, and H. Karl. The gain of overbooking. In E. Frachtenberg and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, volume 5798 of *LNCS*, pages 80–100. 2009.

[5] D. Breitgand, Z. Dubitzky, A. Epstein, A. Glikson, and I. Shapira. Sla-aware resource over-commit in an iaas cloud. In *Proc. of 8th Intl. Conference on Network and Service Management (CNSM)*, pages 73–81, 2012.

[6] W. Cirne and F. Berman. A comprehensive model of the supercomputer workload. In *Proc. of the Intl. Workshop on Workload Characterization*, pages 140–148, 2001.

[7] A. Corradi, M. Fanelli, and L. Foschini. VM Consolidation: a Real Case Based on OpenStack Cloud. *Future Generation Computer Systems*, In Press.

[8] M. Dobber, R. van der Mei, and G. Koole. A prediction method for job runtimes on shared processors: Survey, statistical analysis and new avenues. *Performance Evaluation*, 64(7-8):755–781, 2007.

[9] R. Ghosh and V. K. Naik. Biting off safely more than you can chew: Predictive analytics for resource over-commit in iaas cloud. In *Proc. of 5th Intl. Conference on Cloud Computing*, pages 25–32, 2012.

[10] D. Gmach, J. Rolia, and L. Cherkasova. Selling t-shirts and time shares in the cloud. In *Proc. of Intl. Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 539–546, 2012.

[11] R. Guerin, H. Ahmadi, and M. Naghshineh. Equivalent capacity and its application to bandwidth allocation in high-speed networks. *Selected Areas in Communications*, 9(7):968–981, 1991.

[12] S. He, L. Guo, M. Ghanem, and Y. Guo. Improving resource utilisation in the cloud environment using multivariate probabilistic models. In *Proc. of 5th Intl. Conference on Cloud Computing (CLOUD)*, pages 574–581, 2012.

[13] H. Jin, X. Shi, W. Qiang, and D. Zou. An adaptive meta-scheduler for data intensive applications. *Intl. Journal of Grid and Utility Computing*, 1(1):32–37, 2005.

[14] M. Kalantari and M. K. Akbari. Grid performance prediction using state-space model. *Concurrency and Computation: Practice and Experience*, 21(9):1109–1130, 2009.

[15] Y. C. Lee and A. Y. Zomaya. Energy efficient utilization of resources in cloud computing systems. *The Journal of Supercomputing*, 60(2):268–280, 2012.

[16] libvirt: The virtualization API. Web page at `http://libvirt.org/`, Visited 2013-03-13.

[17] LTTng Project. Linux Trace Toolkit - next generation. Web page at `http://lttng.org/lttng2.0`, Visited 2013-03-13.

[18] X. Meng, C. Isci, J. Kephart, L. Zhang, E. Bouillet, and D. Pendarakis. Efficient resource provisioning in compute clouds via VM multiplexing. In *Proc. of the Intl. Conference on Autonomic Computing (ICAC)*, pages 11–20, 2010.

[19] Nagios - The Industry Standard in IT infrastructure Monitoring. Web page at `http://www.nagios.org/`, Visited 2013-03-13.

[20] Page view statistics for Wikimedia projects. Web page at `http://dumps.wikimedia.org/other/pagecounts-raw/`, Visited 2013-03-13.

[21] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch. Towards understanding heterogeneous clouds at scale:Google trace analysis. Technical Report ISTC–CC–TR–12–101, Carnegie Mellon University, Pittsburgh, PA, USA, Apr. 2012. `http://www.istc-cc.cmu.edu/publications/papers/2012/ISTC-CC-TR-12-101.pdf`.

[22] J. Subramanian, S. Stidham, and C. J. Lautenbacher. Airline yield management with overbooking, cancellations, and no-shows. *Transportation Science*, 33(2):147–167, 1999.

[23] A. Sulistio, K. H. Kim, and R. Buyya. Managing cancellations and no-shows of reservations with overbooking to increase resource revenue. In *Proc. of 8th Intl. Symposium on Cluster Computing and the Grid (CCGrid)*, pages 267–276, 2008.

[24] L. Tomás, A. Caminero, C. Carrión, and B. Caminero. Exponential Smoothing for Network-aware Meta-scheduler in Advance in Grids. In *Proc. of the 6th Intl. Workshop on Scheduling and Resource Management on Parallel and Distributed Systems (SRMPDS)*, pages 323–330, 2010.

[25] L. Tomás, A. C. Caminero, C. Carrión, and B. Caminero. Network-aware meta-scheduling in advance with autonomous self-tuning system. *Future Generation Computer Systems*, 27(5):486 – 497, 2011.

[26] B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource overbooking and application profiling in shared hosting platforms. In *ACM SIGOPS Operating Systems Design and Implementation (OSDI)*, pages 239–254. ACM, 2002.