Postprint

This is the accepted version of a paper presented at 6th Cloud Control Workshop.

Citation for the original published paper:

Tomas, L., Tordsson, J. (2014)
Cloud Service Differentiation in Overbooked Data Centers.

N.B. When citing this work, cite the original published paper.

Permanent link to this version:
http://urn.kb.se/resolve?urn=urn:nbn:se:umu:diva-98010
Cloud Service Differentiation in Overbooked Data Centers

Luis Tomás and Johan Tordsson
Dept. of Computing Science, Umeå University, Umeå, Sweden
{luis,tordsson}@cs.umu.se

Abstract—Low resource utilization in cloud data centers can be mitigated by overbooking but this increases the risk of performance degradation. We propose a three level Quality of Service (QoS) scheme for overbooked cloud data centers to assure high performance QoS for applications that need it. We design a controller that dynamically maps virtual cores to physical cores and whenever feasible shares physical cores among applications. Our evaluation based on real cloud applications and workloads demonstrates that performance isolation can be achieved for critical applications while overall utilization is increased thanks to overbooking.

I. INTRODUCTION

Resource overbooking [1], [2], [3], [4] is a well known technique that can solve or at least mitigate the low resource utilization ratios reported for data centers such as Google [5] and Amazon EC2 [6]. Overbooking is based on accepting and allocating VMs based on real utilization ratios instead of nominal requested capacity, i.e., provisioning more VMs that physical available resources. However, all applications cannot handle the same overbooking level [7] and some none at all. For deadline constrained applications, e.g., computational intensive tasks, average performance is irrelevant as long as the task completes in time. In contrast, interactive applications commonly require a certain minimum performance at all time, e.g., a certain Key Performance Indicator (KPI) such as average throughput or response time.

To address this issue, we extend our previously developed overbooking framework [1] to include Quality of Service (QoS) differentiation in overbooking data centers. At server level, we apply different overbooking ratios for each running VM depending on application needs and behavior. This is implemented through the virtual to physical core pinning functionality provided by the KVM hypervisor. With VM pinning, different isolation levels are provided within the same server, and consequently different overbooking pressure. We offer 3 different QoS levels where either the virtual CPUs (vcpus) are not pinned to the physical cores (pcpus) (Bronze), they are pinned but pcpus can be shared with other vcpus (Silver), or they are pinned with exclusive access to pcpus (Gold).

This exclusive pinning (gold QoS) may affect overall data center utilization as no overbooking is performed for gold cores. To solve this problem we implement an application-aware server pinning controller that allows other applications to use (some of) the exclusive pcpus booked for gold vcpus, if and meanwhile this pcpu sharing does not affect gold application performance at all.

Our experimental results show that QoS differentiation is clearly achieved, where gold level applications are not affected by the overbooking actions. Furthermore, thanks to the pinning controller, the overall utilization ratio is kept high and, even though the gold vcpus are not always being used exclusively, the high quality applications maintain their KPIs, in our case response time or throughput.

II. RELATED WORK

There are several earlier studies of overbooking [8] and its use in cloud data centers. Urgaonkar et al. [7] try to safety overbook cluster resources, guaranteeing applications performance using feedback control. However, they assume that users are capable of providing information regarding the level of overbooking that their applications may tolerate. This information is strongly coupled to the underlying physical infrastructure, as well as the collocated applications. Other works [2] [3] adjust the overbooking ratios in transparent manner to the users. However, in both approaches, unlike our work, once the applications are admitted they do not provide any mechanism to deal with resource shortages due to mispredictions or incorrect overbooking decisions, which affects latency-critical applications.

Furthermore, the suitable overbooking level is unique to each application and depends on how collocated applications interfere each other. An in-depth analysis of collocation problems leading to QoS violations concludes that queuing delays, scheduling delays and load imbalance all significantly impact performance [9]. For this reason, a Borrowed Virtual Time (BVT) scheduler is suggested as replacement for the standard Completely Fair Scheduler (CFS) used in Linux systems. Others try to measure the interference between different applications [10] and make scheduling decisions accordingly [11], [12]. However, it is useful not to only choose which applications to collocate, but also to ensure different degrees of isolation inside a single server. In contrast to these works, our solution is implemented on top of the hypervisor and does not require modifications to the operating system scheduler.

The topic of QoS differentiation based on applications features and requirements has been studied in multiple related fields. For instance, there are several different protocols to provide network QoS differentiation [13], such as Differentiated services (DiffServ) or Integrated Services (IntServ). By using them, the different traffic flows are categorized depending on their requirements and features. This enables a better network
usage and in turns low-latency applications present a better performance. A similar approach in the Grid Computing field was presented by Conejero et al. [14], where three QoS levels are provided by choosing more trustable resources (those which allow advance reservations) for applications that require higher QoS levels, always fulfilling the service level agreements (SLAs) for the high level QoS applications.

Within clouds, there are also works that try to ensure certain QoS through SLAs. For instance, Beloglazov et al. [4], Bobroff et al. [15], and the Sandpiper engine [16] present different methods to detect overload situations and trigger VM migrations to resolve these. However, none of these works provide QoS differentiation and VM migration is also known to affect performance of latency-critical applications [17]. Another related approach is presented by Nguyen et al. [18], where the number of VMs assigned to each cloud application is adaptively adjusted based on medium-term wavelet predictions to ensure Service Level Objectives (SLOs). However they focus on horizontal elasticity, while our focus is on the single VMs behavior.

Another interesting approach is a scheduler that classifies and allocates the incoming applications based on the profiled and expected interferences with other already running VMs [12]. This ensures that interfering applications are allocated in different servers, and thus applications are differentiated and isolated from antagonist applications. As our work provides VM isolation within a single server, the two approaches complement each other.

III. BACKGROUND SCENARIO

Our previous approach to address low utilization ratios at cloud data centers is a framework that increases overall resource utilization by overbooking based on long term risk predictions. Admission control decisions are based on a fuzzy logic risk assessment [19], combined with a proportional-integral-derivative (PID) controller [20]. The PID controller changes the acceptable level of risk over time and the associated overbooking pressure, depending on the deviation of the current data center utilization from the target [1]. In our evaluation we observed that not all applications may handle the same overbooking levels, e.g., one web server application could tolerate much higher target utilization than another one.

Neither the impact of potential overload situations nor the application overbooking tolerance is easy to assess. Therefore, we investigated mitigation and recovery methods for unexpected situations with a scheme that self-optimizes target utilization and adjusts the behavior of running applications if needed [21]. Consequently, we included a short-term mitigation strategy by using Brownout [22], a feedback approach to application performance steering that ensures graceful degradation during load spikes. As a long-term strategy, control information used by Brownout controllers was also sent to the overbooking framework to change the overbooking level (i.e., data center target utilization) depending on how applications behave with current overbooking levels. However, this approach reduces the target utilization of the whole data center

if a single application is having problems, without considering the importance of the application, or if other applications actually can handle even higher levels of overbooking. It could be that with a more efficient use of resources, applications could perform better with no need to reduce the overall utilization.

IV. QOS DIFFERENTIATION MODEL

We extend our overbooking framework [1] to allow different overbooking levels inside the servers by providing QoS differentiation. This QoS differentiation in turn allows to provide better performance, specially for latency sensitive applications, thanks to a higher isolation of their VMs (if needed) through KVM core pinning.

Virtual core to physical core pinning is known to increase application performance [23]. However, in an overbooked scenario, there are more vcpus than available pcpus, therefore the mapping cannot be made in a 1 to 1 basis. Nevertheless, we can still take advantage of the pinning to artificially create separate regions on the server with different overbooking levels, as well as steer the number of pcpus that each vcpu can use, and consequently the number of vcpus sharing the same pcpu (see Figure 1, left mesh). Three different QoS levels are offered by performing the vcpu to pcpu pinning in an application-agnostic manner:

- **Gold**: all application vcpus are pinned to pcpus (in a 1 to 1 basis) and get exclusive access – no other application vcpus can be pinned to these pcpus.
- **Silver**: all application vcpus are pinned to pcpus, but in this case pcpus can be shared with other application vcpus belonging to bronze applications.
- **Bronze**: application vcpus are not pinned to any pcpu and can use any pcpu except the ones booked for gold applications.

These pinning actions are performed following Algorithm 1.

A. Application-Aware Server Pinning Controller

Although Algorithm 1 successfully manages different overbooking ratios inside servers and ensure high performance for those applications that need it (gold VMs), it may impact overall utilization ratios when high quality applications are not efficiently using the capacity they requested, i.e., gold applications are over-provisioned. As the main objective of...
our overbooking framework is to increase resource utilization, providing QoS differentiation should not reduce the utilization ratio. We address over-provisioned gold applications with an application-aware server pinning controller. This simply enables stealing of unused capacity from gold applications, provided that their performance is not affected. It provides performance isolation, but does not rely on dedicated resource allocation to gold applications. In the design of this controller, we move from a black box VM approach to a gray box one using application performance feedback, which has been demonstrated to improve performance in multiple studies [16]. This means that gold applications report how far their KPIs are from the target ones. For this, we use a simple application performance deviation model suggested by Klein et al. [22]. A controller inside the gold applications computes a matching value \( m_i \), that expresses how the application is performing for one of the KPIs:

\[
\begin{align*}
    m_i &= 1 - \frac{r_i}{\bar{r}_i}, \\
    m_i &= 1 - \frac{t_i}{\bar{t}_i},
\end{align*}
\]

where \( r_i \) is the maximum response time over the last control interval and \( \bar{r}_i \) is the target response time, and \( t_i \) and \( \bar{t}_i \) are the minimum and target throughput, respectively. These matching values are positive if the application is able to maintain the desired KPI and become negative if application is suffering performance degradation. Note that this matching value abstracts application performance indicators, such as target response time or throughput, from the infrastructure.

Based on gathered matching value information of the running gold applications, repinning actions are performed by the controller detailed in Algorithm 2, which is based on the following rules:

- **Over-provision**: If a gold VM performs much better than needed (i.e., it is reporting high positive values – over \( T_{\text{high}} \) in Algorithm 2) one of its associated pcpus is included in the list of pcpus that the bronze vcpu can use (see Figure 1, right mesh). While the performance of the gold VM is maintained, this process is repeated until all the pcpus are shared. This increases utilization and bronze applications throughput without impacting gold applications performance.

- **Slightly over-provision**: If a gold VM performs as it should, but without larger margins (i.e., reported matching values are all positives but with small minimum values – over \( T_{\text{low}} \) in Algorithm 2) the current pin mapping is maintained.

- **Under-provision**: If gold VM is close or already have some problems to maintain target KPI (i.e., some negative or close to 0 matching values are reported – below \( T_{\text{low}} \) in Algorithm 2) all the pcpus associated to this VM are again used exclusively.

In the rest of work, we use a conservative sharing scheme with \( T_{\text{high}} = 0.4 \) and \( T_{\text{low}} = 0.2 \), i.e. gold applications only share their pcpus when their KPIs are being fulfilled by far (high value for \( T_{\text{high}} \)), and gets them back even before noticing any problems (positive value for \( T_{\text{low}} \)). This is due to the objective of our proposed QoS differentiation scheme - to guarantee KPIs when possible, and always for gold applications, while at the same time maintaining the good overall utilization ratio achieved thanks to overbooking [1]. With other objectives, less conservative schemes can be used (lower values of \( T_{\text{high}} \) and/or negative values for \( T_{\text{low}} \)). In addition, less restrictive policies could be tried, e.g., sharing more than one core at the time and/or returning half of the cores at a time. Further analysis of such variations of the overall algorithm is however outside the scope of this paper.
A. Applications and Workload

To create a representative cloud environment, we mix different VMs that can be grouped in two classes following the "boulders and sand" VMs behavior reported in [5]. In this model, the boulders are represented by interactive applications running in large VMs, for a long time (months, years), and accessed by a varying number of users over time – usually presenting some seasonality in their access pattern (e.g., more users during working hours than during nights). For this application class, we used two popular cloud benchmarks: RUBiS and RUBBoS. RUBiS [24] is an auction website benchmark modeled after eBay, whilst RUBBoS [25] is a bulletin board benchmark modeled after Slashdot. In these experiments the boulders outline the duration of our evaluations. Therefore we consider a fixed set of them in each run: 2 VMs (RUBiS + RUBBoS) requesting half of the server capacity (8 vcpus and 14 GB memory each). For both of them, a workload consisting of a number of queries received as a function of time is generated using information extracted from the Wikipedia [26] traces. For each service, we selected a day and time-shifted the original workload 12 hours for RUBiS, as shown in Figure 2. This creates different trends and peaks, and represents that not all services may have the same daily usage patterns. The client queries were generated using the httpmon tool\(^1\).

On the other hand, the sand VMs (non-interactive applications) consist of relatively short lived and non periodic applications, e.g. VMs running computational tasks, with highly heterogeneous and time-varying resource requirements [27]. For this VM class we have modeled two kind of behaviors applicable to each VM dimension - bursty and steady. We have made use of the 3node test from GRASP benchmarks [28] (for the steady CPU behavior) and several shell scripts to generate burstiness in the different capacity dimensions. Additionally, to measure the performance achieved over time for the sand applications, we have used an application that continuously solves random sudokus\(^2\) and reports the throughput achieved over time. The sand applications arrival pattern is generated using a Poisson distribution with \(\lambda = 20\) seconds.

B. Performance evaluation

In order to evaluate the applications performance in an overbooking scenario and the advantages of our QoS differentiation proposal, we measure the KPIs for both application classes:

- The average and 95-percentile response time for the interactive applications (boulders) – RUBiS and RUBBoS.
- The average and accumulated throughput for the non-interactive applications (sand) – Sudoku solver.

Additionally, we want to demonstrate that overall utilization achieved thanks to overbooking is not impacted due to the service differentiation performed. Therefore, we evaluate the overall capacity allocated (the sum of applications’ resource requests that quantifies the provider’s revenue) and the real utilization achieved (here, CPU usage as it is the saturated dimension in these experiments).

Figures 3, 4 and 5 show the same executions for the different QoS levels and our service pinning controller (Gold-Stealing), where RUBiS, RUBBoS and Sudoku tasks are running concurrently. Figure 3 shows the response times, server utilization, and capacity allocated for RUBiS. The black dashed straight line represent the target utilization (75%) as well as the target high limit for the response time (500ms). It shows that the higher the QoS, the shorter the response times (both average and 95th percentile), but at the expense of lower overall utilization and capacity allocated. For the Gold-Stealing scenario (Figure 3d) similar response times to the gold QoS level are achieved. Unlike the gold scenario, utilization and capacity allocation rates are pretty close to the ones achieved by the bronze or silver QoS level – utilization falls less than 3% on average, mainly due to the last period.

Figure 4 shows the same information but for RUBBoS. As before, response times become shorter as we increase the QoS level, but overall utilization decreases. As RUBBoS response time is not as linear with number of user requests as RUBiS is, response times are not always acceptable with Bronze and Silver QoS, with the current level of overbooking (pursuing 75% server utilization). This is depicted in Figure 4a and Figure 4b after minute 1000, when response times start to be longer and longer until they are no longer acceptable. By contrast, thanks to the gold stealer controller, the system takes advantage of the periods of time where RUBBoS application does not need that much capacity (i.e., no need of total core isolation), and increases overall utilization by letting other applications use pcpus associated to the RUBBoS VM. Then, when RUBBoS has more concurrent users to serve (from

\(^1\)https://github.com/cloud-control/httpmon
\(^2\)http://norvig.com/sudoku.html
minute 1000 onwards, see Figure 2), these cpus are again only used for the RUBBoS VM, therefore the utilization drops during that period, but the agreed performance is fulfilled.

Finally, Figure 5 shows the aggregated (average of total number of sudokus solved by all concurrent sudoku tasks per minute) and average (average number of sudokus solved per task per minute) throughput achieved by the sudoku tasks when RUBiS and RUBBoS are executed with Bronze, Silver, Gold and Gold-Stealing QoS, respectively. Regarding aggregated throughput, although gold-stealer has more fluctuations (larger standard deviation), the overall aggregated throughput is quite close to the one achieved by bronze and silver scenarios but with remarkably improved response times for the interactive applications. If compared to the gold scenario, the aggregated throughput achieved is remarkably increased, without any important performance degradation for the interactive services. There are no large differences regarding average throughput per sudoku VM. Gold scenarios show a bit more fluctuations, but even with a bit higher overall throughput per VM. By using the Wilcoxon statistical test [29], a non-parametric statistical hypothesis test that compares two related samples to assess whether their population mean ranks differ, we conclude that there is no evidence of average throughput drop for gold-stealer (large p-value: 0.5769), while it is statistically significant that gold-stealer outperforms gold average throughput (small p-value: 3.422e-06).

To sum up, utilization ratios achieved by the gold-stealing technique are really good, pretty close to bronze and silver, but also ensures QoS provided to interactive jobs and with a similar aggregated and average throughput. A summary of the different tests is presented at Table I. It also includes another case where the target utilization is increased to 85% for the Gold scenario. The twofold objective is to: (1) show that QoS isolation is maintained even for excessive overbooking ratios; and (2) that pursuing too high utilization ratios is not beneficial, as previously evaluated [1] – although utilization increases, the aggregated throughput does not increase and the average throughput decreases.

VI. CONCLUSIONS

Overbooking is used as a technique to improve the low utilization ratios at todays cloud data centers. However, it also increases the risk of performance degradation. Furthermore, not all applications can deal with the same overbooking...
levels and even some of them cannot afford overbooking at all. To address both issues (ensuring QoS performance and high resource utilization) at the same time, we propose a QoS differentiation mechanism within overbooked cloud data centers, allowing different overbooking ratios at servers level. By using KVM pinning we create three different QoS levels, where either the VMs are pinned to specific cores and use them exclusively, they are pinned but shared with other VMs, or they are not pinned at all.

Although this QoS schema achieves performance differentiation and the assurance objective, it has an impact at the overall resource utilization in the presence of gold VMs. To deal with this problem we implemented a controller that steals some CPU time from gold applications if and only if they are not affected by these actions. This controller provides service differentiation and performance assurance without impacting the overall resource utilization.

We plan to extend the isolation to not only pcpus, but also considering I/O and memory. A better isolation at the three dimensions will provide a more predictable performance, and in turns will lead to better QoS differentiation and assurance.

**References**


**TABLE I: Summary of the tests. Values in bold highlight deficiencies of a given technique.**

<table>
<thead>
<tr>
<th>QoS</th>
<th>Service</th>
<th>Response Time (ms)</th>
<th>Utilization (%)</th>
<th>Throughput (Sudokus/sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Avg.</td>
<td>Max. of 95th</td>
<td>Avg.</td>
</tr>
<tr>
<td>Bronze (75%)</td>
<td>RUBiS</td>
<td>66.5</td>
<td>493.61</td>
<td>73.04</td>
</tr>
<tr>
<td></td>
<td>RUBBoS</td>
<td>79.46</td>
<td>2919.12</td>
<td>62.35</td>
</tr>
<tr>
<td>Silver (75%)</td>
<td>RUBiS</td>
<td>55.8</td>
<td>394.81</td>
<td>72.79</td>
</tr>
<tr>
<td></td>
<td>RUBBoS</td>
<td>163.33</td>
<td>2954.04</td>
<td>69.86</td>
</tr>
<tr>
<td>Gold (75%)</td>
<td>RUBiS</td>
<td>26.73</td>
<td>139.2</td>
<td>62.37</td>
</tr>
<tr>
<td></td>
<td>RUBBoS</td>
<td>25.17</td>
<td>241.85</td>
<td>68.85</td>
</tr>
<tr>
<td>GoldStealer (75%)</td>
<td>RUBiS</td>
<td>37.45</td>
<td>256.41</td>
<td>69.86</td>
</tr>
<tr>
<td></td>
<td>RUBBoS</td>
<td>38.32</td>
<td>237.43</td>
<td>68.85</td>
</tr>
</tbody>
</table>

**TABLE II: Summary of the tests. Values in bold highlight deficiencies of a given technique.**

<table>
<thead>
<tr>
<th>Test</th>
<th>Response Time (ms)</th>
<th>Utilization (%)</th>
<th>Throughput (Sudokus/sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Avg.</td>
<td>Max. of 95th</td>
<td>Avg.</td>
</tr>
<tr>
<td>GoldStealer (75%)</td>
<td>RUBiS</td>
<td>35.99</td>
<td>327.27</td>
</tr>
</tbody>
</table>