Service Level Management for Iterative Pre-Copy Live Migration

Thomas Treutner, Helmut Hlavacs
Research Group Entertainment Computing
University of Vienna, Austria
Email: firstname.lastname@univie.ac.at

***Abstract*—The ability to live migrate virtual machines (VMs) between physical servers without any perceivable service interruption is pivotal for building more energy efficient Cloud Computing infrastructures in the future. Nevertheless, energy efficiency is not worth the effort if quality metrics (e.g., QoS, QoE) are severely decreased by, e.g., dynamic consolidation using live migration. In this work, we present results for a low level approach by patching KVM to implement Xen's live migration stop conditions, and a high level approach by monitoring the progress and service level state of a live migration and estimating the impact on the service level during the remaining migration time. We compare these approaches with an unpatched, vanilla KVM version and different sets of parameters used for live migration. Our service level management approaches offer superior QoS during migration. Especially, they allow to migrate also highly utilized VMs with comparably small influence on QoS.**

I. Introduction

The vast majority of servers in classical data centers is underutilized for a significant amount of time [2]. These servers operate at a very low rate of efficiency and waste huge amounts of energy. Concerns regarding efficiency are one of several important reasons why modern data centers frequently use system virtualization. Live migration is an important mechanism offered by virtualization. It provides the possibility to establish a more energy efficient target distribution of virtual machines, as it can be used for dynamic consolidation (turning off servers) and scaling (using more or faster servers). We consider the ability to live migrate VMs between physical servers without any perceivable service interruption to be pivotal for building more energy efficient computing infrastructures in the future. It enables us to dynamically shift workload between servers with regard to a cost model, where energy costs of a computing infrastructure play an important role.

However, only little work has been done yet to study the influence of live migration on service quality and especially to ensure that live migration does not lead to service level violations. In this work, we extend our previous efforts [8] by evaluating different implementations of iterative pre-copy live migration and their effects on web service levels. The key problem addressed in this work is that iterative pre-copy live migration relies on page transfer rates being significantly higher than page dirtying rates. Then, converging to a small amount of remaining dirty pages is feasible within short time and small influence on QoS. However, highly utilized VMs often show page dirtying rates close to current standard Gbit Ethernet transfer rates. Then, the migration process will converge only slowly or not at all, increasing the resource pressure onto the VM for long period of time. As a consequence, the VM's services and its users experience unacceptably low QoS. Due to lost revenue, this could render energy savings gained from live migration void. We state our contribution as follows:

1) We empirically examine the QoS during migration at CPU utilization levels from very low to almost 100%. We show results for an unpatched vanilla KVM version and different sets of parameters (bandwidth, downtime).
2) We present a low level approach using a small patch implementing Xen's stop conditions for KVM. We demonstrate by experiment that Xen's implementation of live migration is superior to KVM regarding QoS.
3) In a high level approach, we monitor the migration's progress and estimate its remaining time. In contrast to related work, we do not need to modify the hypervisor. We estimate the service level at the migration's prospected end and pro-actively take evasive action.
4) Both low and high level approaches offer significantly better QoS during migration, especially for highly utilized VMs. For extremely high utilizations, the high level approach performs best.

The remainder of the paper is structured as follows: In Section II, we discuss related research efforts. In Section III, we briefly describe our experimental infrastructure. In Section IV, we present our approaches to ensure QoS during migration. In Section V, we present results and in Section VI, we draw conclusions and give outlooks on future work.

II. Related Work

The ability to live migrate a virtual machine first described by Clark et al. [4] for the `Xen` hypervisor. `Xen`'s live migration algorithm works iteratively and is an important building block for many other hypervisor's live migration mechanisms. The crucial assumption is that the amount of dirty pages quickly converges to a minimum, and service interruptions are imperceptible. If this assumption of convergence does not hold, the live migration may run indefinitely, and stop conditions are necessary to ensure that the process does finish eventually. In Xen, three stop conditions [1] are defined:

1) Less than fifty pages were dirtied during the last iteration
2) 29 pre-copy iterations have been carried out

3) More than three times the amount of memory the VM has allocated was transferred over the network

`qemu-kvm`'s implementation [9] of live migration is similar, but there is an important difference: Dirty memory pages are iteratively transferred over network until the remaining amount can be transferred within a predefined (but adjustable) time limit. In contrast to [9], where two stop conditions are described, the actual implementation has no stop conditions at all. If the rate of dirtying pages is too high and/or the time limit is too low, the migration process takes forever and the administrator or a management application has to increase the time limit or cancel the process.

Hirofuchi et al. have implemented post-copy live migration [5], [6] for KVM and have shown [7] its potential, especially regarding energy-efficiency. Their implementation is publicly available on the project page[1], but the source code is marked obsolete. We are looking forward to extend our experiments once there is a stable code base.

Akoush et al. [1] have investigated parameters affecting the time required for a live migration, especially page dirtying rate and link speed. Their efforts result in models able to predict the migration time and downtime within 90% accuracy for synthetic and real-world benchmarks. Nevertheless, their approach requires tracking of dirty pages, which makes resource intensive low level adaptions necessary within the hypervisor. Among other benchmarks, the authors have employed SPECweb, but have not studied quality of service.

Breitgand et al. [3] have examined a possible trade-off between the service downtime and the migration delay. They have examined in-band live migration, where the same network link is used for migration and workload. Obviously, during live migration these two facilities compete for resources and contentions are likely. The authors have carried out a simulation using real traces, using the portion of requests not served with a maximum response time as a cost function. Although simulated and calculated results are similar, an empirical proof was not presented. They come to the conclusion that quality of service during live migration is depending on the available network bandwidth. Although this result is comprehensible, the underlying scenario and assumptions are rather exotic. Even today, several networks exist in typical data centers. It is reasonable to assume that live migration traffic is carried by a dedicated network. Furthermore, scanning the memory address space for dirty pages is a CPU intensive process.

Voorsluys et al. [11] have evaluated the cost of VM live migration for a two-tier Web 2.0 scenario. They come to the conclusion that live migration overhead is "acceptable but can not be disregarded" [11]. The authors only consider a fully loaded VM. We suggest that different workload situations should be considered, leading to a more differentiated view.

Liu et al. [10] have presented a model able to predict the time an iterative pre-copy live migration takes to complete, based on measuring the page transfer rate and the page dirtying rate. Information about dirty pages is gathered by adapting the Xen hypervisor. They have evaluated migration delay, downtime, network traffic and energy consumption for various applications running inside a migrated VM, e.g., SPECweb2005. However, quality of service metrics, e.g., response times and service levels, are not part of their evaluation.

[1]http://grivon.apgrid.org/quick-kvm-migration

III. Experimental Setup

A. Infrastructure

For carrying out our experiments, we have used a similar infrastructure as decribed in [8] . We have updated our software stack to Debian Squeeze AMD64, `qemu-kvm-1.0`, `Linux 3.2.0`. All servers involved have two GBit Ethernet network interfaces plugged into separate GBit switches. The first is used exclusively for workload (here, HTTP and SQL traffic), the second for storage, management and live migration traffic. Thus, we do not perform in-band migration, so workload and live migration traffic do not compete for bandwidth. A central node is sending HTTP workload at varying intensities to the VM and tracking its service level. The maximum allowed response time is 1.5 s. For a detailled description, refer to [8] .

B. Migration Progress Logging

During the live migration, we track its progress every second using `libvirt`, which provides macroscopic progress information. Most important are 1) the amount of memory transferred over network, denoted as *processed memory*, which is not to be mistaken for the amount of clean memory pages, and 2) the amount of remaining memory, in the following defined as *dirty pages*. The iterative pre-copy live migration mechanism copies *dirty* pages to the destination host and marks them to be *clean*, while the VM is still running. Thus, pages can be written concurrently and become *dirty* again, therefore they must be copied over the network again in the next iteration. Based on these two progress metrics, we estimate the *dirty page rate*, which we define as the amount of memory that changed its status from clean to dirty per second during the interval i, by

$$D_i = \frac{(p_i - p_{i-1}) - (d_{i-1} - d_i)}{t_i}, \tag{1}$$

where p_i is the amount of processed memory at the end of interval i, p_{i-1} the respective amount at the end of interval $i-1$, d_i and d_{i-1} are the respective amounts of dirty memory, and t_i is the length of interval i in seconds. Note that the amount of processed memory is continuously growing throughout the whole migration with every page that is transferred over the network. Further, we assume $d_i < d_{i-1}$, meaning that the process is at least slowly converging. The basic idea of our approach is to compare the increase of processed memory with the decrease of dirty pages. If no memory page that was already transferred over the network would be written again by the VM, then

$$\begin{aligned} p_i - p_{i-1} &= d_{i-1} - d_i & (2) \\ (p_i - p_{i-1}) - (d_{i-1} - d_i) &= 0 & (3) \end{aligned}$$

as each processed page would perfectly equate to a dirty page less. However, the VM is active during all but the last pre-copy iteration and can cause memory pages change their state from clean to dirty. Thus, the increase of processed memory does not equate to the decrease of dirty memory, yielding

$$(p_i - p_{i-1}) - (d_{i-1} - d_i) > 0, \tag{4}$$

and we use this difference to estimate the amount of memory that changed its state from clean to dirty in the last interval. The alternative to this estimation would be to modify the hypervisor to frequently dump a huge bit vector signaling the state of each memory page and to analyse their development over time. We also calculate the transfer rate within each interval i,

$$R_i = \frac{(p_i - p_{i-1})}{t_i}, \tag{5}$$

which is simply the amount memory transferred in the interval i. Naturally, if the page dirtying rate is getting close to the transfer rate, then the live migration process is not progressing, as the amount of dirty pages is not converging towards zero.

IV. Service Level Management

A. Xenalike KVM

In a low level approach, we have developed a small patch[2] for KVM to implement Xen's live migration stop conditions. The final iteration is forced when either

1) the VM's memory address space was scanned 29 times for dirty pages, meaning 29 iterations were carried out,
2) more than three times of the VM's allocated memory was transferred.

Then, independently from the amount of remaining dirty pages, the VM is paused and the final iteration starts. Subsequently, the VM is unpaused at the destination host. In between, the VM's services are of course unavailable, possibly affecting service levels. However, if a longer downtime prevents that a non-converging migration puts more and more resource pressure onto the VM, the impact on response times and service levels could be actually positive.

B. Service Level Managed KVM

Although the patch implementing Xen's stop conditions in KVM is very small, we were finally interested in evaluating a high level approach. As mentioned above, Liu et al. [10] have presented a simple model able to predict the amount of time an iterative pre-copy live migration takes to complete, based on the page transfer and dirtying rates. We have first tested if the model is valid for our experimental setup. The measured migration delay and the model are visible in Figure 1, showing very small residuals. Hence, we can predict the migration delay without low level modifications as used in [10] .

[2]http://lists.gnu.org/archive/html/qemu-devel/2011-09/msg01754.html

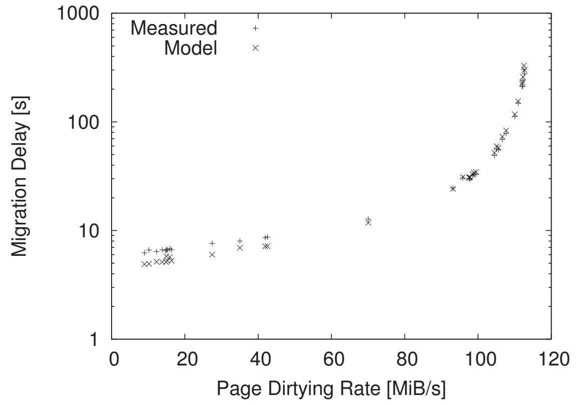

Fig. 1. Migration Delay modelled based on the Page Dirtying Rate

We have extended our monitoring facility by a *service level management component*. Every second, we have predicted the remaining time the migration will take to complete. We have defined a *service level target* of 80% for the migration process. We have defined an *action threshold* and a corresponding *counter*. If the counter reaches the threshold value, here 3, the migration is forced. As the monitoring facility is active every second, we regard a value of 3 as a trade-off between robustness and quick action. Events incrementing the counter are: 1) The dirty page rate is greater than the transfer rate, and the remaining time is not computable. The migration will not converge, not even slowly. 2) If the current response time is greater than the allowed maximum, we assume it will not decrease during the rest of the migration, meaning that all requests during the remaining estimated migration time will violate the maximum allowed response time too. From that, we calculate the service level at the end of the migration, and if it is violating the service level target, the action counter is incremented.

V. Results

In the following, we discuss the results found in our experiments. For all experiments, our discussion is based on Figure 2, Figure 3 and Figure 4.

a) Unpatched KVM: Unpatched KVM is unable to migrate within acceptable time if the page dirtying rate comes close to 32 MiB/s, with catastrophic impact on the service level. The reason is that KVM by default migrates with a hard-coded limit of 32 MiB/s. Hence, the level of utilization where the migration is unable to converge is reached quickly. The default downtime target is 30 ms, actual service downtimes are an order of magnitude bigger. The experiment was aborted at a CPU utilization of 40%, with a service level below 50%, and migrations taking half an hour.

b) Bandwidth Increased KVM: Increasing the bandwidth to 1 Gbit/s, deferes the system's breaking point, but the development is similar. Page dirtying rates closer to 1 Gbit/s or 125 MiB/s would further increase the time required until the migration has converged to a minimum of dirty pages. The migration takes minutes at high utilization and results in highly varying QoS, as visible in Figure 4.

c) Bandwidth Increased KVM, Strict Downtime: Decreasing the maximum allowed downtime to 4 ms leads to actual downtimes of approximately 0.3 s. However, even with a migration speed of 1 Gbit/s, the system is unable to converge to a state where transferring the remaining dirty pages would take such a short time, even when the VM only has to bear medium workload intensity. The experiment was aborted at a CPU utilization of 35%, with a service level of 0%, as the migration thrashed the VM in question. Clearly, there is a need for a systematic trade-off between migration delay and downtime, offering acceptable QoS during migration for all intensities of utilization.

d) Xenalike KVM: Our patched version of KVM implementing Xen's stop conditions, is also set to 4 ms of maximum allowed downtime, again resulting in approximately 0.3 s true service downtime. In contrast to the experiments described above, the true downtime is increased from approximately 0.3 s to 0.8 s-1.2 s when the workload intensity, and consequently, the page dirtying rate, rises. The stop conditions force the migration's final iteration, causing a slightly higher downtime, but significantly decreasing the migration delay and its negative impact on response times and QoS. There is a slow and linear decrease of service levels while the CPU utilization rises until a level of approximately 75%. Subsequently, the service level decreases non-linear, but the system is always able to finish the migration. Instead of clinging firmly to an unachievable low downtime target, resulting in disproportionally long migration delays and catastrophic service levels, the final precopy iteration is forced if the system does not converge within a reasonable amount of time and effort.

e) Service Level Managed KVM: Our high level approach was also set to 4 ms of maximum allowed downtime, showing the same results as for Xenalike discussed above. The service level managed KVM approach achieves similar service levels as Xenalike. For very high CPU utilization above 90%, there is a small advantage compared to Xenalike. Due to a service level that is already too low, the migration is forced without further monitoring, resulting in downtimes of 5 s.

VI. Conclusions and Future Work

In this work, we have evaluated the influence of live migration on service levels for a web server scenario. We have shown that the default implementation and settings are clearly giving room for improvement. Beginning at a CPU utilization of approximately 35%, migration leads to catastrophic service levels. Increasing the migration bandwidth just defers the system's breaking point, but offers no general solution. Both of our approaches result in significantly better service levels and prevent the VM from thrashing and complete service disruption. They are achieving acceptable service levels during live migration also for higher workload intensities by trading off migration delay and migration downtime. The service downtime increases in favour of shorter migration delays, gaining significantly better service levels. As a result, they enable service providers to migrate also during phases of high workload intensity without causing disproportional high service level degradation. For future work, we are especially interested in adapting our experiments regarding web service QoS to implementations of post-copy live migration.

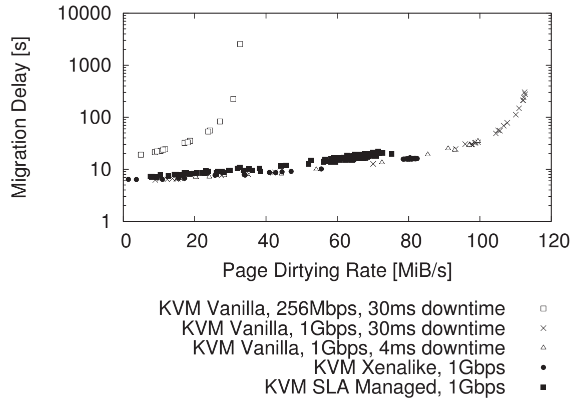

Fig. 2. Influence of the rate of page dirtying on the time required to complete a live migration. Note the logarithmic scaling on the y-axis.

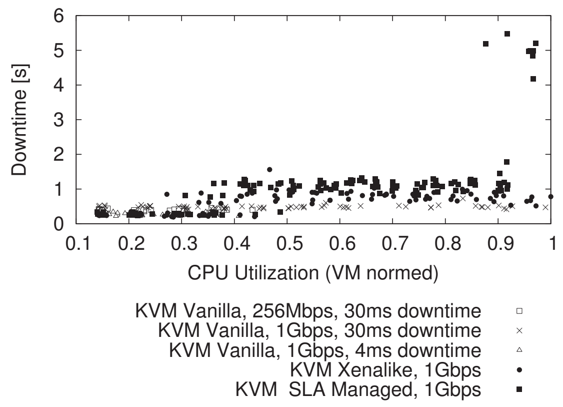

Fig. 3. Comparison of the service downtime development with increasing CPU utilization.

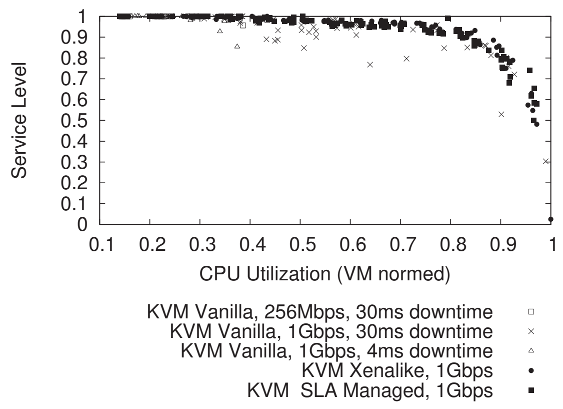

Fig. 4. Comparison of the service level development with increasing CPU utilization.

References

[1] S. Akoush, R. Sohan, A. Rice, A. Moore, and A. Hopper. Predicting the performance of virtual machine migration. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on*, pages 37–46. IEEE, 2010.

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

[3] D. Breitgand, G. Kutiel, and D. Raz. Cost-aware live migration of services in the cloud. In *Proceedings of the 3rd Annual Haifa Experimental Systems Conference*, page 11. ACM, 2010.

[4] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 273–286, Berkeley, CA, USA, 2005. USENIX Association.

[5] M. R. Hines, U. Deshpande, and K. Gopalan. Post-copy live migration of virtual machines. *SIGOPS Oper. Syst. Rev.*, 43(3):14–26, 2009.

[6] M. R. Hines and K. Gopalan. Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. In *VEE '09: Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 51–60, New York, NY, USA, 2009. ACM.

[7] T. Hirofuchi, H. Nakada, S. Itoh, and S. Sekiguchi. Making vm consolidation more energy-efficient by postcopy live migration. In *CLOUD COMPUTING 2011, The Second International Conference on Cloud Computing, GRIDs, and Virtualization*, pages 195–204, 2011.

[8] H. Hlavacs and T. Treutner. Predicting web service levels during vm live migrations. In *Systems and Virtualization Management (SVM), 2011 5th International DMTF Academic Alliance Workshop on*, pages 1–10. IEEE, 2011.

[9] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. kvm: the linux virtual machine monitor. In *Proceedings of the Linux Symposium*, volume 1, pages 225–230, 2007.

[10] H. Liu, C.-Z. Xu, H. Jin, J. Gong, and X. Liao. Performance and energy modeling for live migration of virtual machines. In *Proceedings of the 20th international symposium on High performance distributed computing*, HPDC '11, pages 171–182, New York, NY, USA, 2011. ACM.

[11] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya. Cost of virtual machine live migration in clouds: A performance evaluation. In *CloudCom '09: Proceedings of the 1st International Conference on Cloud Computing*, pages 254–265, Berlin, Heidelberg, 2009. Springer-Verlag.