Real-Time Issues in Live Migration of Virtual Machines*

Fabio Checconi[1], Tommaso Cucinotta[1], and Manuel Stein[2]

[1] Scuola Superiore Sant'Anna, Pisa, Italy
[2] Alcatel-Lucent Bell Labs, Stuttgart, Germany

Abstract. This paper addresses the issue of how to meet the strict timing constraints of (soft) real-time virtualized applications while the Virtual Machine (VM) hosting them is undergoing a live migration. To this purpose, it is essential that the resource requirements of a migration are identified in advance, that appropriate resources are reserved to the process, and that multiple VMs sharing the same resources are temporally isolated from each other. The first issue is dealt with by introducing a stochastic model for the migration process. The other ones by introducing a methodology making use of proper scheduling algorithms (for both CPU and network) that allow for reserving resource shares to individual VMs. Also, an extensive set of simulations have been done by using traces of a VLC video server virtualized by using KVM on Linux. The traces have been obtained by patching KVM at the kernel level, and the same patch constitutes an important step towards the complete implementation of the proposed technique. The obtained results highlight the benefits of the proposed approach.

1 Introduction

Virtualization technology is gaining more and more interest in the high-performance computing world. However, its use implies a level of sharing of the available hardware resources that is unprecedented. In fact, now it is possible to share the same physical host (PH) across multiple concurrently running OS instances, and it is possible to live-migrate an entire OS to a different PH if needed, with very limited service interruption times. Unfortunately applications may suffer from the concurrent access to shared resources, like CPUs, network links and storage, if proper allocation policies are not adopted. This is of particular relevance whenever the hosted applications exhibit real-time/QoS requirements. This kind of constraints is in place not only for interactive applications but also for batch activities whose QoS levels need to adhere to precise service-level agreements.

One key issue about guaranteeing proper QoS levels is how to keep control of what exactly happens during the live migration of a VM. For example, the

* The research leading to these results has been supported by the European Commission under grant agreement n.214777, in the context of the IRMOS Project. More information at: http://www.irmosproject.eu.

time needed to migrate a VM from one PH to another may be highly variable depending on the network load that is being generated by other VMs on the subnet, as well as on the computational load that is being generated by other VMs concurrently running on the migration start and end PHs.

Contributions of this Paper. This paper addresses the issue of how to guarantee that the process of VM live migration exhibits a temporal behavior that may be kept under control. This problem is faced with by investigating the resource requirements needed by the migration process, and by proposing mechanisms that may be used to guarantee appropriate resource availability when the live migration occurs. To this purpose, a theoretical framework is introduced for estimating the variability of the duration of the overall migration time and of the down-time, with respect to the page migration policy. Also, the issue of how to guarantee that the migration process itself does not interfere too much with the other running VMs is addressed.

2 Background

Migration is the process used to transfer a VM from the host on which it resides to a different one. A migration is said to be *live* if the execution of the VM is not interrupted during most of the transfer.

As described in [1], when a Virtual Machine Monitor (VMM) has to migrate a VM to a new PH, all the resources associated with the VM have to be transferred to the new host, comprising the memory, the internal state of the devices and of the virtual CPU. The most time-consuming resource to transfer is generally the memory. in several ways, and This paper focuses on the so-called *pre-copy* of memory pages, where the VMM begins transferring the pages while the VM is running, checks what pages are dirtied again after the transfer, and retransmits them. This process is repeated for a certain number of times, after which the VM is stopped, the remaining dirty pages are transmitted and the VM is started on the destination.

In these systems, migration of storage is not usually a concern, because of the use of network-based storage solutions that allow for a client VM to seamlessly keep accessing the data after migration to a different host.

Related Work. The use of migration in virtualized environments is a well known subject in the literature. The first proposed mechanisms just stopped VMs on the source PH, saving their state, and restarting them using the saved state on the remote PH (see, e.g., [2]). These approaches suffer of long down times, that often are not acceptable when VMs are executing interactive applications, and furthermore make it impossible to keep the VM connections active.

To overcome these limitations, live migration has been introduced, where the transfer of the VM memory is done while the VM is running. *Pre-copy* migration was introduced in [3], the VM is restarted at and later used in NomadBIOS [4], a VMM built on top of the L4 microkernel [5], and then in Xen [6].

With *demand migration*, derived from the *copy-on-reference* mechanism described in [7], memory is migrated to the destination after the VM has restarted

its execution remotely. Recently, in [8], the authors introduced *post-copy* migration, which works enhancing the demand-migration approach by reordering the page transfers for the purpose of minimizing the time spent by the VM waiting for transfers after it restarts execution. With *self-migration*, introduced in [9], the guest OS has awareness of being executed inside a VM, and exploits standard checkpointing techniques to transfer the memory and internal state of the OS by itself.

In [10], a comparison between classical static VM allocation and dynamic resource reallocation enabled by live migration is done. The authors of [11] introduced an analytical model of VM migration to estimate the expected improvement of the hosted service's response time due to a migration decision.

The issues of temporal isolation across multiple VMs concurrently running on the same host, and of how to run real-time virtualized tasks under such conditions, have been considered in [12] and [13] by (partially) the same authors of this paper, but live-migration has not been addressed yet.

3 Live Migration Model

This section presents a stochastic model for real-time migration, whose purpose is twofold: on one hand, it allows for identifying the requirements of the live migration process; on the other hand, it constitutes a motivation for the ordering of pages that are used in the process, as it will be detailed in Section 5.

A VM is characterized by a set of memory pages $\{p_1, \ldots, p_N\}$ assumed for simplicity to be of fixed size equal to P bytes. Assume the bandwidth available for the transfer is constant and equal to b bytes per second (this is possible by using the techniques described in Section 4), and let T denote the time interval needed to transfer a single page $T = \frac{P+H}{b}$ (under the assumption that no compression is used), where H is the overhead in bytes introduced by the migration protocol for each page. Assume that, for the time horizon spanning the entire migration process, each page p_i has a constant probability π_i of being accessed at least once for writing within each time frame T, and assume the events of write access for each page are all independent from one another. Assume the migration process works according to the following steps:

1. at time t_1 in which the migration starts, the set of pages $\mathcal{D}_1$ to be transmitted is set to the entire set of pages used by the VM; let n_1 denote its cardinality $n_1 = |\mathcal{D}_1|$;
2. for $k = 1, \ldots, K$, repeat the following: all the n_k pages in $\mathcal{D}_k$ ($n_k = |\mathcal{D}_k|$) are transferred, with a bandwidth of b bytes per second, according to the order specified by the function $\phi_k : \{1 \ldots n_k\} \to \{1 \ldots N\}$ (i.e., the pages are transmitted in the order $p_{\phi_k(1)}, \ldots, p_{\phi_k(n_k)}$); the transfer ends at $t_{k+1} = t_k + n_k T$, in which n_{k+1} pages $\mathcal{D}_{k+1}$ are found to have become dirty again;
3. stop the VM and transfer the last n_{K+1} pages in $\mathcal{D}_{K+1}$, up to the migration finishing time $t_f = t_{K+1} + n_{K+1}\frac{P+H}{b_d}$, using a bandwidth of b_d bytes per second, with $b_d \geq b$.

Then, the crucial values characterizing the migration process are the down-time $t_d = t_f - t_K$ during which the VM is stopped, and the overall migration time $t_{tot} = t_f - t_1$, which may now be expressed in terms of the other quantities introduced above:

$$t_d = \left(\frac{P+H}{b_d}\right) n_{K+1}, \quad t_{tot} = \left(\frac{P+H}{b}\right) \sum_{k=1}^{K} n_k + t_d. \tag{1}$$

The above introduced notation and assumptions are at the basis of the following results, that focus on the case $K = 1$ for the sake of brevity. All proofs are omitted but they are available at: http://retis.sssup.it/ tommaso/vhpc09-proofs.pdf.

Proposition 1 *The probability of a page p_i that is not dirty at time t_1 to become dirty and thus need to be transmitted in the final migration round is:*

$$\Pr\{p_i \in \mathcal{D}_2 \mid p_i \notin \mathcal{D}_1\} = 1 - (1 - \pi_i)^{n_1}. \tag{2}$$

Proposition 2 *The probability of a page p_i that is dirty at time t_1 to become dirty again and thus need to be transmitted in the final migration round is:*

$$\Pr\{p_i \in \mathcal{D}_2 \mid p_i \in \mathcal{D}_1\} = 1 - (1 - \pi_i)^{n_1+1-\phi_1^{-1}(i)}, \tag{3}$$

where $\phi_1^{-1}(\cdot) : \{1 \dots N\} \to \{1 \dots n_1\}$ denotes the inverse of the $\phi_1(\cdot)$ function.

Theorem 1 *The expected overall migration time (with $K = 1$) is:*

$$E[t_{tot}] = \left(\frac{P+H}{b}\right) n_1 + \left(\frac{P+H}{b_d}\right) \left[n_1 - \sum_{j=1}^{n_1} \left(1 - \pi_{\phi_1(j)}\right)^{n_1+1-j} + (N - n_1) - \sum_{i \notin \mathcal{D}_1} (1 - \pi_i)^{n_1} \right]. \tag{4}$$

Theorem 2 *The order $(\phi_k(1), \dots, \phi_k(n_k))$ of transmission of the pages that minimizes the expected number of dirty pages found at the end of the k^{th} live migration step must satisfy the following condition:*

$$\forall j \; \pi_{\phi_k(j)} (1 - \pi_{\phi_k(j)})^{n_k - j} \leq \pi_{\phi_k(j+1)} (1 - \pi_{\phi_k(j+1)})^{n_k - j}. \tag{5}$$

Corollary 1 *If the probabilities π_i are all lower than $\frac{1}{n_k+1}$, than the optimum ordering is obtained for increasing values of the probabilities π_i. On the other hand, if the probabilities are all greater than $\frac{1}{2}$, then the optimum ordering is obtained for decreasing values of the π_i.*

Reducing the Overall Migration Time. Among the pages to be transmitted there are pages which are accessed by the VM with a very high frequency. For example, pages containing data used by the guest OS for scheduling processes are found as always written at each observation instant. Therefore, it is not

convenient to transmit such pages at each migration step, because these pages would need to be transmitted again in the last step, when the VM is stopped. Therefore, it is possible to modify the migration algorithm as follows: among the n_k pages that are found as dirty at start of step k, for a set of pages $F_k \subset \mathcal{D}_k$ delay the transmission to when the VM is stopped. One possibility is to choose the pages for which the access probabilities are higher than a threshold value $\overline{\pi}$: $F_k \triangleq \{p_i \in \mathcal{D}_k \mid \pi_i \geq \overline{\pi}\}$. Focusing on $K = 1$, the following holds:

Proposition 3 *If the transmission of the pages* $F_1 \triangleq \{p_i \in \mathcal{D}_1 \mid \pi_i \geq \overline{\pi}\}$ *is delayed, then the new overall transmission time* $\tilde{t}_{tot}$ *and the new down-time* $\tilde{t}_d$ *satisfy the following:*

$$E\left[\tilde{t}_d\right] \leq E\left[t_d\right] + |F_1|\left(1-\overline{\pi}\right)\left(\frac{P+H}{b_d}\right)$$
$$E\left[\tilde{t}_{tot}\right] \leq E\left[t_{tot}\right] - |F_1|\left(\frac{P+H}{b}\right) + |F_1|\left(1-\overline{\pi}\right)\left(\frac{P+H}{b_d}\right).$$

Therefore, with a sufficiently low $\overline{\pi}$, it is possible to achieve a negligible increase in the expected down-time but with a substantial decrease of the overall migration duration. Simulations shown in Section 5 will confirm this.

Practical Implications. The theoretical framework introduced above relies on precise knowledge of the probability π_i of access of a VM to each page p_i. From a practical perspective, these probabilities are not actually known, however it is possible to estimate their values at run-time. One possible way of doing this is by sampling periodically, with a fixed period what pages in the VM have been tagged as dirty by the kernel, then resetting the dirty flag bits and repeating the measure several times.

Building an accurate ordering of dirty pages satisfying Equation 5 may lead to non-negligible overheads (actually, a possible algorithm for doing this is still being developed). Therefore, this paper proposes a couple of algorithms: a simple LRU based approach, where the pages that are transmitted first during each live migration step are the least recently used ones, and a frequency-based approach, where the pages are transmitted in order of increasing access frequency. The results section will show results obtained with the introduced orderings.

4 Real-Time Issues

In order to keep control over the live migration process, so as to achieve a predictable timing behavior of the process, the technique proposed in this paper foresees the adoption of proper scheduling strategies for the involved resources, namely computation and network resources.

Computation Resources. In order to guarantee proper processor shares to Virtual Machines that are concurrently running on the same Physical Host, it is possible to exploit scheduling strategies that are available for the Linux kernel as separate patches in the domain of soft real-time applications.

For example, the AQuoSA scheduler [14] developed in the context of the FRESCOR [3] European Project provides a user-space API and a set of command-line tools that allow for providing to a thread, process or group of them, a scheduling guarantee by the kernel. Such guarantee is expressed in terms of a pair (Q, P), with the meaning that within each period of duration P microseconds, the thread (or thread group) is reserved the CPU for Q microseconds. This guarantee has a strong theoretical foundation, as the scheduler has been written as a variation of the Constant Bandwidth Server, a real-time scheduler whose description can be found in [15].

Also the POSIX Sporadic Server [16] and the framework presented in [17], both recently developed in the context of the IRMOS European Project [4] may be used for the purpose of temporal isolation among multiple processes running on the same Linux OS.

Such mechanisms may be used in order to encapsulate each VM (along with all the threads it is composed of) within a proper scheduling guarantee, whose parameters need to be found by using appropriate benchmarking techniques.

Network Resources. For the estimation of the overall number of page transmissions during the live migration to be accurate, the network bandwidth b should be of a constant available bitrate. With a fixed size of P bytes per page, the time slot T for a page to become dirty has a hyperbolic dependence on the bandwidth b and so is any variation ΔT. Note, that the average probability π_i of a page being accessed in a single slot T increases if the network bandwidth decreases. Further, the recursive nature of the iterative process accounts for an autocorrelated contribution of T to the overall number of required page retransmissions during each iteration.

Practically, the longer a page transmission takes, the higher is the probability of pages being dirtied which increases the number of required retransmissions. With more pages being dirtied, the duration of the follow-up iteration increases during which more pages can be dirtied again. ΔT propagates analogously, because a single variation of the time required for a page transmission results in an autocorrelated propagation of error in the estimation model.

In a managed network computing cluster, the available bandwidth for a possible migration can be properly reserved. An example for an innovative infrastructure that considers network resource isolation and reservation under multi-hop conditions for cloud computing is the Intelligent Service-Oriented Network Infrastructure (ISONI) of the IRMOS project. For the assumptions of a constant duration T per page transmission to hold, a constant bitrate scheduling algorithm is recommended to determine whether the required overall migration time and expected VM downtime are acceptable for the efficiency of the infrastructure and the SLA of the VM, respectively. Further research would be required to account for the effect and the allowable degree of bandwidth variations during the migration mechanism.

[3] More information at the URL: `http://www.frescor.org`.
[4] More information at the URL: `http://www.irmosproject.eu`.

5 Evaluation Results

Implementation. In order to support the algorithms described in this paper, the basic infrastructure has been implemented modifying the KVM hypervisor and the Linux kernel itself. This comprises a page tracing mechanism [5] that exposes to user-space the set of pages that have been accessed in write (dirtied) within each observation interval. Page accesses are traced using a bitmap inside the hypervisor; every time a writable mapping is created by the guest, the hypervisor sets the bitmap position corresponding to the newly mapped page. Periodically the bitmap is zeroed and all the writable mappings are reset to read-only. Currently, a simple user-space program saves this information to trace files, which have been used for the simulations shown later. A complete implementation of the live-migration mechanisms proposed in this paper would just require to exploit this information to affect the transmission order of the pages in the existing KVM migration code.

Simulations. This section reports simulation results that prove effectiveness of the proposed technique in reducing both the overall migration time and the down-time. Simulations rely upon traces gathered from real virtualized applications running on KVM on Linux, patched with the tracer previously described in this section.

A virtualized VideoLAN Client (VLC) server has been chosen as the target application scenario for the evaluation of the proposed technique. This scenario raises interesting real-time issues (concerning the down time during a live migration) in the case of live streaming of a video acquired from a camera and transmitted in real-time to the viewers, or in the case of a video streaming service that needs a low seeking latency (as needed during an interactive distributed video editing session). A few traces have been collected independently on VMs that were running the just cited application, while a few clients were accessing the provided service. The observation period has been set to $250ms$. In the collected trace, the VM had a set of about 6500 mapped pages (with 16 KBytes per page) when the migration was simulated. This would correspond to an service interruption of about 8 seconds, if a stop-and-transfer were performed with a bandwidth of 100 MBit/sec (plus the protocol overheads) reserved to the process.

Figure 1 shows the results obtained by simulating the live migration process, in terms of achieved number of down-pages, transmitted while the VM is stopped, and overall number of transmitted pages during the entire migration. These results have been measured for various number of live migration steps/rounds, which correspond to the different points on each curve. Also, multiple page transmission policies have been used in the simulation of the same live migrations, corresponding to the different curves on the figure. The `Simple` curve refers to the standard address-based ordering of pages, which is used by default by the current version of KVM, the `LRU` curve is obtained when transmit-

[5] The tracing patch used to collect the data used in the experimental section is available at `http://feanor.sssup.it/~fabio/linux/kvm/page-trace/`

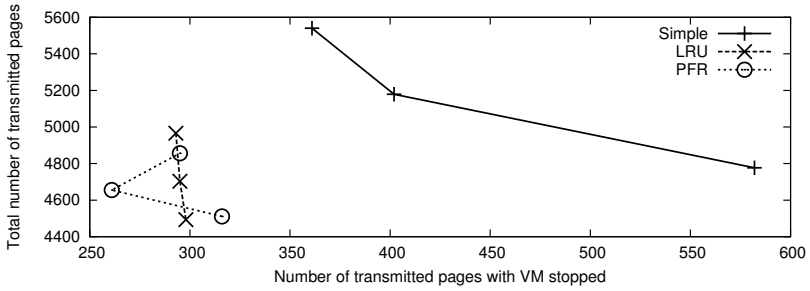

Fig. 1. Performance achieved at varying values of K (various points within each curve) and page migration policy (various curves).

ting first the Least Recently Used pages, and finally the `PFR` curve is obtained when transmitting first the pages which have been dynamically measured (at run-time) to have the lowest write access frequency. The bandwidth guaranteed to the migration process has been set to 50 Mbit/s.

The rightmost points correspond to $K = 1$ and the leftmost points correspond to $K = 3$. Increasing K, generally leads to an increase of the overall transmission time, but also to a significant reduction in the down-time. For example, with the standard KVM policy, with a single round there are 400 pages left to transmit when the VM is stopped, whilst with 3 rounds these pages are reduced to about 270 (roughly a 48% down-time reduction).

Comparing the rightmost, leftmost and middle points on each curve, it is evident how a simple LRU reordering of the pages transmitted during a live migration, as compared to the default address-based order, can achieve a great reduction in both the down-time and the overall migration time: the pages left to transmit when the VM is stopped can decrease from about 570 down to about 300 (47% down-time reduction) when $K = 1$, or a decrease from about 360 to about 290 (19.4% reduction) with $K = 3$. Also, the overall number of pages to transmit during the migration is reduced from about 4800 down to 4500 (6.25% reduction) with $K = 1$, or from about 5500 down to about 5000 (9.1% reduction).

The figure also highlights that a further down-time decrease may be obtained by transmitting the pages in increasing order of write access probability (PFR curve), however its relevance needs to be compared with the additional overhead of keeping an exact ordering by frequency of access, as compared to the simple LRU ordering, which may be implemented by a list that is manipulated in $O(1)$.

Figure 2 reports a similar plot obtained when an LRU page transmission policy has been used, with the additional trick to delay transmission of the most frequently accessed pages. The various curves in the figure correspond to different values of the threshold value $\overline{\pi}$ (indicated in the legend) for the page access probability, over which the page transmission has been delayed (as described in Section 3). The further improvements achieved by this technique are evident from the figure, where the down-pages may be further reduced from

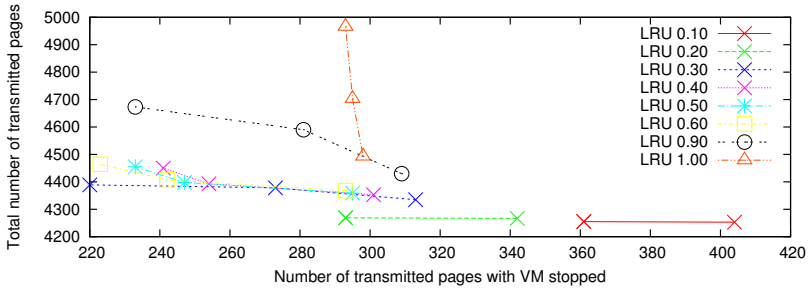

Fig. 2. Performance achieved at varying values of K (various points within each curve) and of $\overline{\pi}$ (various curves).

300 down to 220 (27% reduction), and the overall time from 5000 down to 4400 (12% decrease) when using a $\overline{\pi} = 0.30$.

6 Conclusions

This paper presented a technique for live migration of real-time virtualized applications. The achievement of very low and predictable down times, as well as the availability of guaranteed resources during the migration process, are key factors for obtaining outages in the provided service with a negligible impact.

A probabilistic model of the migration process has been introduced, for the purpose of building a sound mathematical theory over which to found a novel set of migration policies, such as the proposed one based on a simple LRU order, or the more complex one based on the observed page access frequencies in the past VM history. The LRU policy has been proved (by simulation) to achieve a good degree of effectiveness in decreasing the down-time and overall migration time, still keeping an acceptable level of overhead.

However, the proposed technique needs a deeper evaluation over the full implementation that is being developed, especially on the side of the possible trade-offs between accuracy in gathering the information needed for optimizing the page transmission order, the corresponding run-time overhead, and possible variations to the page transmission schemes adopted.

References

1. Clark, C., Fraser, K., Hand, S., Hansen, J.G., Jul, E., Limpach, C., Pratt, I., Warfield, A.: Live migration of virtual machines. In: NSDI'05: Proceedings of the 2nd Symposium on Networked Systems Design & Implementation, Berkeley, CA, USA, USENIX Association (2005) 273–286
2. Kozuch, M., Satyanarayanan, M.: Internet suspend/resume. In: WMCSA '02: Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications, Washington, DC, USA, IEEE Computer Society (2002) 40

3. Theimer, M.M., Lantz, K.A., Cheriton, D.R.: Preemptable remote execution facilities for the v-system. SIGOPS Oper. Syst. Rev. **19**(5) (1985) 2–12
4. Hansen, J.G., Henriksen, A.K.: Nomadic operating systems. Master's thesis, Dept. of Computer Science, University of Copenhagen, Denmark (2002)
5. Härtig, H., Hohmuth, M., Liedtke, J., Shönberg, S., Wolter, J.: The performance of micro-kernel-based systems. In: SOSP '97: Proceedings of the Sixteenth ACM Symposium on Operating System Principles, ACM (1997) 66–77
6. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: SOSP '03: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, New York, NY, USA, ACM (2003) 164–177
7. Zayas, E.: Attacking the process migration bottleneck. SIGOPS Oper. Syst. Rev. **21**(5) (1987) 13–24
8. Hines, M.R., Gopalan, K.: 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, New York, NY, USA, ACM (2009) 51–60
9. Hansen, J.G., Jul, E.: Self-migration of operating systems. In: EW11: Proceedings of the 11th workshop on ACM SIGOPS European workshop, New York, NY, USA, ACM (2004) 23
10. Kochut, A.: On impact of dynamic virtual machine reallocation on data center efficiency. In: MASCOTS '08: Proceedings of the 2008 16th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems. (Sept. 2008) 1–8
11. Kochut, A., Beaty, K.: On strategies for dynamic resource management in virtualized server environments. In: MASCOTS '07: Proceedings of the 2007 15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, Washington, DC, USA, IEEE Computer Society (2007) 193–200
12. Cucinotta, T., Anastasi, G., Abeni, L.: Real-time virtual machines. In: Proceedings of the 29^{th} IEEE Real-Time System Symposium (RTSS 2008) – Work in Progress Session, Barcelona (December 2008)
13. Cucinotta, T., Anastasi, G., Abeni, L.: Respecting temporal constraints in virtualised services. In: To appear in Proceedings of the 2^{nd} IEEE International Workshop on Real-Time Service-Oriented Architecture and Applications (RTSOAA 2009), Seattle, Washington (July 2009)
14. Palopoli, L., Cucinotta, T., Marzario, L., Lipari, G.: AQuoSA — adaptive quality of service architecture. Software – Practice and Experience **39**(1) (2009) 1–31
15. Abeni, L., Buttazzo, G.: Integrating multimedia applications in hard real-time systems. In: Proceedings of the IEEE Real-Time Systems Symposium, Madrid, Spain (December 1998)
16. Faggioli, D., Mancina, A., Checconi, F., Lipari, G.: Design and implementation of a POSIX compliant sporadic server. In: Proceedings of the 10^{th} Real-Time Linux Workshop (RTLW), Mexico (October 2008)
17. Checconi, F., Cucinotta, T., Faggioli, D., Lipari, G.: Hierarchical multiprocessor cpu reservations for the linux kernel. In: Proceedings of the 5^{th} International Workshop on Operatin g Systems Platforms for Embedded Real-Time Applications (OSPERT 2009), Dublin, Republic of Ireland (July 2009)

Appendix: Proofs

Proposition (Was Proposition 1 in the text). *The probability of a page p_i that is not dirty at time t_1 to become dirty and thus need to be transmitted in the final migration round is:*

$$\Pr\{p_i \in \mathcal{D}_2 \mid p_i \notin \mathcal{D}_1\} = 1 - (1 - \pi_i)^{n_1}. \tag{6}$$

Proof. The probability may be written as the complement of the probability of the page never being accessed during the transmission of the n_1 pages. Let $E_{i,j}$ denote the probability that p_i is not accessed during the transmission of the j^{th} page, with $j = 1, \ldots, n_1$. Then, due to the assumptions on the page access probabilities, the $\{E_{i,j}\}_{j=1,\ldots,n_1}$ events are all independent, therefore: $\Pr\{p_i \in \mathcal{D}_2 \mid p_i \notin \mathcal{D}_1\} = 1 - \Pr\{E_{i,1} \wedge \ldots \wedge E_{i,n_1}\} = 1 - \prod_{j=1}^{n_1} \Pr\{E_{i,j}\} = 1 - (1 - \pi_i)^{n_1}$. □

Proposition (Was Proposition 2 in the text). *The probability of a page p_i that is dirty at time t_1 to become dirty again and thus need to be transmitted in the final migration round is:*

$$\Pr\{p_i \in \mathcal{D}_2 \mid p_i \in \mathcal{D}_1\} = 1 - (1 - \pi_i)^{n_1 + 1 - \phi_1^{-1}(i)}, \tag{7}$$

where $\phi_1^{-1}(\cdot) : \{1 \ldots N\} \to \{1 \ldots n_1\}$ denotes the inverse of the $\phi_1(\cdot)$ function.

Proof. The probability may be written as the complement of the probability of the page never being accessed in the period going from the time the page starts to be transmitted $t_1 + \phi_1^{-1}(i)T$, to the time the transmission round is over $t_1 + n_1 T$, corresponding to a total of $n_1 + 1 - \phi_1^{-1}(i)$ time slots of duration T, i.e., $\Pr\{p_i \in \mathcal{D}_2 \mid p_i \in \mathcal{D}_1\} = 1 - \Pr\left\{E_{i,\phi_1^{-1}(i)} \wedge \ldots \wedge E_{i,n_1}\right\} = 1 - \prod_{j=\phi_1^{-1}(i)}^{n_1} \Pr\{E_{i,j}\} = 1 - (1 - \pi_i)^{n_1 + 1 - \phi_1^{-1}(i)}$. □

Theorem (Was Theorem 1 in the text). *The expected overall migration time (with $K = 1$) is:*

$$E[t_{tot}] = \left(\frac{P+H}{b}\right) n_1 + \left(\frac{P+H}{b_d}\right) \left[n_1 - \sum_{i \in \mathcal{D}_1} (1 - \pi_1)^{n_1 + 1 - \phi_1^{-1}(i)} + (N - n_1) - \sum_{i \notin \mathcal{D}_1} (1 - \pi_i)^{n_1} \right]. \tag{8}$$

Proof. From Equation 1, $t_{tot} = \left(\frac{P+H}{b}\right) n_1 + t_d$, where the first term is constant and known, and the expected down-time $E[t_d]$ may be computed as follows. Let X_i be a stochastic variable equal to 1 if the page p_i is dirty and needs to be transmitted in the next step, and 0 otherwise. Clearly, the number of pages n_2 that are dirty and thus need to be transmitted in the next step, is equal to: $n_2 = \sum_{i=1}^{N} X_i$. However, for each page $i \in \mathcal{D}_1$, the probability that $X_i = 1$ is the probability that the page becomes dirty again in the time-interval

$\left[t_1 + \phi_1^{-1}(i)T,\, t_1 + n_1 T\right]$, which is computed by means of Equation 7. On the other hand, for each page $i \notin \mathcal{D}_1$, the probability that $X_i = 1$ is the probability that the page becomes dirty in the time-interval $[t_1,\, t_1 + n_1 T]$, computed by using Equation 6. Therefore:

$$E[n_2] = E\left[\sum_{i=1}^{N} X_i\right] = \sum_{i \in \mathcal{D}_1} E[X_i] + \sum_{i \notin \mathcal{D}_1} E[X_i] \tag{9}$$

$$= \sum_{i \in \mathcal{D}_1} \left[1 - (1-\pi_i)^{n_1+1-\phi_1^{-1}(i)}\right] + \sum_{i \notin \mathcal{D}_1} \left[1 - (1-\pi_i)^{n_1}\right] \tag{10}$$

$$= n_1 - \sum_{i \in \mathcal{D}_1} (1-\pi_i)^{n_1+1-\phi_1^{-1}(i)} + (N - n_1) - \sum_{i \notin \mathcal{D}_1} (1-\pi_i)^{n_1} \tag{11}$$

The proof is easily obtained by considering that $E[t_d] = \left(\frac{P+H}{b_d}\right) E[n_2]$. □

Theorem (Was Theorem 2 in the text). *The order $(\phi_k(1), \ldots, \phi_k(n_k))$ of transmission of the pages that minimizes the expected number of dirty pages found at the end of the k^{th} live migration step must satisfy the following condition:*

$$\forall j\, \pi_{\phi_k(j)}(1-\pi_{\phi_k(j)})^{n_k-j} \leq \pi_{\phi_k(j+1)}(1-\pi_{\phi_k(j+1)})^{n_k-j}. \tag{12}$$

Proof. During the k^{th} algorithm step, n_k pages are transmitted in the order $(\phi_k(1), \ldots, \phi_k(n_k))$. Therefore, the j^{th} transmitted page $p_{\phi_k(j)}$ has $n_k - j + 1$ time slots for becoming dirty again before the next step. The probability of the page becoming dirty again between the time in which it starts to be transmitted, and the time in which the step finishes, is: $1 - \left(1 - \pi_{\phi_k(j)}\right)^{n_k+1-j}$. Now, let X_j be a stochastic variable equal to 1 if the page $p_{\phi_k(j)}$ gets dirty again and needs retransmission in the next step, and 0 otherwise. Clearly, the number N_k of pages that, after being transmitted during step k, become dirty again and need to be retransmitted in the next step, is equal to: $N_k = \sum_{j=1}^{n_k} X_k$. As a consequence, the expected value of N_k may be written as:

$$E[N_k] = \sum_{j=1}^{n_k} E[X_j] = \sum_{j=1}^{n_k} \phi_{\phi_k(j)}(n_k - j + 1) = n_k - \sum_{j=1}^{n_k} \left(1 - \pi_{\phi_k(j)}\right)^{n_k-j+1}. \tag{13}$$

At this point, the theorem proof is easily obtained by absurd. Suppose the minimum value of $E[N_k]$ is obtained when the pages do not respect the ordering in 12. Then, there exists an index j such that Condition 12 does not hold. The contribution to $E[N_k]$ due to the two pages is $2 - \left(1 - \pi_{\phi_k(j)}\right)^{n_k-j+1} - \left(1 - \pi_{\phi_k(j+1)}\right)^{n_k-(j+1)+1}$, while swapping the two pages into the sequence would lead to a contribution of $2 - \left(1 - \pi_{\phi_k(j)}\right)^{n_k-(j+1)+1} - \left(1 - \pi_{\phi_k(j+1)}\right)^{n_k-j+1}$, leaving the contributions due to the other pages unchanged. This, under the assumption of $\pi_{\phi_k(j)}\left(1 - \pi_{\phi_k(j)}\right)^{n_k-j} > \pi_{\phi_k(j+1)}\left(1 - \pi_{\phi_k(j+1)}\right)^{n_k-j}$, corresponds to a *decrease* of $E[N_k]$, leading to a contradiction. □

Corollary (Was Corollary 1 in the text). *If the probabilities π_i are all lower than $\frac{1}{n_k+1}$, than the optimum ordering is obtained for increasing values of the probabilities π_i. On the other hand, if the probabilities are all greater than $\frac{1}{2}$, then the optimum ordering is obtained for decreasing values of the π_i.*

Proof. In the first case, for all positive exponents $n_k - j$, the function $\pi(1-\pi)^{n_k-j}$ is always monotonically increasing in π, therefore Condition 12 is equivalent to having increasing probabilities $\pi_{\phi_k(i)}$ in the sequence $\left\{\pi_{\phi_k(1)}, \ldots, \pi_{\phi_k(n_k)}\right\}$. Similarly, in the first case, the function $\pi(1-\pi)^{n_k-j}$ is monotonically decreasing in π. Therefore, the same condition translates to requiring decreasing values of $\pi_{\phi_k(i)}$ in the sequence $\left\{\pi_{\phi_k(1)}, \ldots, \pi_{\phi_k(n_k)}\right\}$. □

Proposition (Was Proposition 3 in the text). *If the transmission of the pages $F_1 \triangleq \{p_i \in \mathcal{D}_1 \mid \pi_i \geq \overline{\pi}\}$ is delayed, then the new overall transmission time $\tilde{t}_{tot}$ and the new down-time $\tilde{t}_d$ satisfy the following:*

$$E\left[\tilde{t}_d\right] \leq E\left[t_d\right] + |F_1|\left(1-\overline{\pi}\right)\left(\frac{P+H}{b_d}\right)$$
$$E\left[\tilde{t}_{tot}\right] \leq E\left[t_{tot}\right] - |F_1|\left(\frac{P+H}{b}\right) + |F_1|\left(1-\overline{\pi}\right)\left(\frac{P+H}{b_d}\right).$$

Proof. If these pages were transmitted, then $E[|F_{k+1}|]$ would be equal to $|F_k| - \sum_{i\in F_k} \overline{\pi}_i^{n_k - j_k(i)+1}$, where $j_k(i)$ is the transmission position of the page i. This may be upper bounded by $E\left[|F_{k+1}|\right] \geq |F_k| - \sum_{i\in F_k}\left(1-\tilde{\pi}\right) = |F_k|\,\tilde{\pi}$. Similarly, at the end of the last step (the K^{th}), the number of pages in F_1 that would have to be retransmitted would be $|F_k|\left[1-\left(1-\tilde{\pi}\right)^K\right]$. So, if these pages are not transmitted at the k^{th} step, then a contribution of $\sum_{k=1}^{K}|F_1|\left[1-\left(1-\tilde{\pi}\right)^k\right]$ pages is removed from the expected number of pages to be transmitted before stopping the VM, but a contribution of $|F_1|\left(1-\tilde{\pi}\right)$ would be added to the expected number of pages to be transmitted while the VM is stopped. □