Live Migration of Virtualized Edge Networks: Analytical Modeling and Performance Evaluation

Franco Callegati, Walter Cerroni
D.E.I. "G. Marconi" - University of Bologna
via Venezia 52, 47521 Cesena - Italy
E-mail: name.surname@unibo.it

***Abstract*—Following the current evolution of virtualization techniques and software defined networking, edge networks might evolve towards a fully virtualized implementation by means of a number of virtual machines working cooperatively to perform the tasks of existing network middleboxes. In such a scenario the possibility to migrate groups of cooperating virtual machines as a whole set may be a very important feature, but what will be the performance issues of this solution? The live migration performance of multiple virtual machines working in some sort of correlated manner is a topic that has not been widely studied in the literature. This manuscript presents a model reasonably simple to implement that may be used to derive some performance indicators such as the whole service downtime and the total migration time. The model is used to compare some scheduling strategies for the migration and provide guidelines to such implementation.**

***Index Terms*—Virtualization, Software Defined Networks, Analytical modeling, Performance evaluation.**

I. Introduction

In current networks IP lost its simple and effective end-to-end delivery paradigm. Networking services require a number of additional and more complex functionalities, typically stateful, which cannot be supported by IP and must be performed by middleboxes operating at logical layers above routing and forwarding (L3). This has been outlined very clearly by some recent investigations. The results in [1] show that the amount of middleboxes and the burden related to running them is considered a major issue by network administrators, whereas [2] shows that at least 25% of the TCP flows analyzed suffered of some form of interference by middleboxes running tasks at or above layer 4.

In public networks, middleboxes management is an issue for operators [3], especially at the edge where the evolution of terminals is bringing a degree of complexity in the implementation of services never experienced before. Emerging technologies, such as Software Defined Networking (SDN) [4] and Network Function Virtualization (NFV) [5], offer a new opportunity to migrate the middleboxes in a virtualized environment, with possible very significant advantages.

In essence, these key enablers will allow the development of network nodes based on standard off-the-shelf hardware, cheap but powerful enough to run virtualized network functions and services. Edge networks will then be made of inexpensive nodes and user devices capable of collapsing the OSI layers (e.g., L2 to L7) on standard hardware solutions. In this scenario edge networks will become a distributed environment made of clouds of virtual resources (operated even by diverse players) interconnected by a simpler and less hierarchical core network [6].

Such a radical transformation will enable new roles and business opportunities, completely reshaping the value chains in the entire Telco-ICT world. One of the main challenges behind this vision is the capability of dynamically instantiating, orchestrating and migrating multiple Virtual Machines (VMs) running network services and applications across the providers transport networks. Ensembles of VMs will be strictly related and intertwined to implement sets of virtual resources and services that network operators, and even users, must be able to configure and program.

This important topic has still to be deeply investigated, since in the past mostly the migration of single VMs has been subject of research studies. In this work we intend to address this gap. In Section II we briefly review the issue of VM migration, outlining that the *pre-copy* migration strategy is the one that apparently better matches the specific application described above. Then in Section III we propose an extension to known analytical models for the migration of a single VM to the case of multiple VMs to be migrated with some form of coordination and discuss the two obvious alternatives of serial and parallel migration. In Section IV performance evaluation results obtained with the model are presented, comparing the two migration strategies and discussing some quantitative engineering guidelines. Finally in Section V some conclusions are drawn.

II. Live Migration of Virtual Machines

The problem of migrating virtual machines live, i.e. while the VM is running and end-user processes are active, is well known and has already been addressed in the literature. It is impossible to migrate the VM without stopping it even for a short time, therefore the migration impacts the running services. As a consequence a major performance parameter to be considered is the so called *downtime* (T_{down} in the following), i.e. the amount of time the VM will be stopped because of the migration. On the other hand the *total migration time* (T_{mig} in the following) is another important parameter that measures the impact that the migration of the VM has on the whole IT infrastructure, since the resources used during the migration phase cannot be used for other tasks.

There are several strategies for the migration of VMs that aim at optimizing one performance indicator or another according to the overall system requirements. For the purpose of this study we will focus on a typical *pre-copy* strategy [7], which combines a *push* phase, during which the VM memory is transferred in subsequent rounds while the VM is still running, and a *stop-and-copy* phase, during which the VM is stopped and just a residual part of the memory is transferred.

This migration strategy can be implemented if the virtualization manager enables some form of shadow memory that makes possible to understand which part of the VM memory was modified in a given time window. This is the case in typical modern systems such as Xen and KVM [8].

The push phase consists of n memory transfers also called *rounds*. At round 0 the whole memory of the VM is copied to the new location. This will take time and, since the VM is running, part of the memory will be modified by the running processes. Assuming that the modified memory pages are less than the total, the transfer is iteratively repeated and during round i the memory pages that were modified during round $i-1$ are transferred. If memory pages *dirtied* during each round tends to decrease the duration of the rounds will also decrease and at some stage the estimated number of pages to be transferred, as well as the duration of round n, will be reasonably low. At this point round n is performed in the stop-and-copy phase: the VM is stopped, the modified memory pages are transferred and the VM can be consistently restarted at the new location.

A model for the migration of a single VM with pre-copy strategy is proposed in [9]. The model is presented for a single VM and using some simplifying conditions, such as for instance a constant memory page dirtying rate. The model is used to evaluate the downtime as well as the overall transfer time for the VM. We start from these results and extend them as explained in the following section.

III. Modeling Multiple VM Migration

In this paper we intend to tackle the more complex problem of migrating groups of correlated VMs, therefore the performance parameters to be considered will depend on what happens to the migration of several and not just one VM. The extension of the model in [9] to the case of a set of VMs is not trivial, since it depends on how the multiple VMs are scheduled for the live transfer. Furthermore, moving a set of mutually dependent VMs requires a new definition of service downtime, and this again is strictly related to the role and migration timing of each VM.

Consider the general case of a set of M VMs, where $V_{\mathrm{m},j}$ is the amount of memory allotted to the j-th VM in the set, $D_j(t)$ its memory page dirtying rate and P_j its memory page size. According to the pre-copy migration algorithm, when the j-th VM transfer starts at time $t_{0,j}$, the whole memory $V_{\mathrm{m},j}$ is copied to the destination host while the VM instance is still running at the source host. Then, the transfer of modified memory pages is iterated n_j times at $t_{1,j},\ldots,t_{n_j,j}$, until either the amount of dirty memory is below a given threshold V_{th} or the maximum number of iterations $n_{\max}$ is reached.

Let $R_j(t)$ be the bit rate used to transfer the j-th VM to the destination host and $V_{i,j}$ the amount of dirty memory to be copied during the i-th round. The following equations describe the iterative migration process:

$$V_{0,j} = V_{\mathrm{m},j} = \int_{t_{0,j}}^{t_{1,j}} R_j(t)dt \tag{1}$$

$$V_{i,j} = \int_{t_{i-1,j}}^{t_{i,j}} P_j D_j(t)dt = \int_{t_{i,j}}^{t_{i+1,j}} R_j(t)dt, \quad i=1,\ldots,n_j \tag{2}$$

As a first approximation, we can assume that both $R_j(t)$ and $D_j(t)$ are constant during the whole migration process. For the transfer rate this is a reasonable assumption, considering that some form of bandwidth sharing mechanisms with guaranteed bit rate must be enforced in order to keep the performance of the virtualized edge network under control. As for the memory page dirtying rate, its behavior strongly depends on the nature of the applications running on the VM: however, as a first approximation we can consider that the amount of pages being modified by the running processes is proportional to their execution time [9].

Then we also assume that the M VMs in the set are allotted the same amount of memory and that the applications running on them show the same page dirtying rate. We are aware that this may not be completely true in the virtualized edge network architecture we are considering, since the memory profile of each VM strongly depends on the specific network functions it has to perform. However, these assumptions allow to simplify the equations and to capture the macroscopic performance aspects of multiple VM live migration, which is the main purpose of this paper. A more refined model is left for future work.

Based on the previous assumptions, we have

$$D_j(t) = D, \quad V_{\mathrm{m},j} = V_{\mathrm{m}}, \quad P_j = P, \quad \forall j = 1,\ldots,M$$

whereas $R_j(t) = R_j$ depends on how the M VMs are scheduled to be migrated. Calling $T_{i,j} = t_{i+1,j} - t_{i,j},\ i=0,\ldots,n_j$, equations (1) and (2) become

$$\begin{aligned} V_{0,j} &= V_{\mathrm{m}} = R_j\, T_{0,j} \\ V_{i,j} &= P\, D\, T_{i-1,j} = R_j\, T_{i,j}, \quad i=1,\ldots,n_j \end{aligned}$$

from which we can compute the duration of the i-th iteration as

$$T_{i,j} = V_{\mathrm{m}}\,\frac{(P\,D)^i}{R_j^{i+1}}, \quad i=0,\ldots,n_j \tag{3}$$

The pre-copy migration algorithm is sustainable as long as the average memory dirtying rate is smaller than the transfer rate, i.e. when $\lambda_j = (P\,D)/R_j < 1$. The last round (stop-and-copy phase) will happen either for the smallest i such that

$$V_{i,j} = \lambda_j^i\, V_{\mathrm{m}} \le V_{\mathrm{th}}$$

or when $n_{\max}$ iterations are reached. Therefore, the number of iterations and the time needed to migrate the j-th VM are

$$n_j = \min\left\{\left\lceil \log_{\lambda_j} V_{\rm th}/V_{\rm m} \right\rceil, n_{\max}\right\} \quad (4)$$

$$T_{{\rm mig},j} = \sum_{i=0}^{n_j} T_{i,j} = \frac{V_{\rm m}}{R_j}\,\frac{1-\lambda_j^{n_j+1}}{1-\lambda_j} \quad (5)$$

The computation of the total migration time and downtime of a set of VMs strictly depends on how and when the VMs are scheduled to be migrated. In the following we consider two simple cases: (i) when the M VMs are transferred one at a time (*serial migration*); (ii) when all the M VMs are transferred simultaneously (*parallel migration*). We then compare the performance of these two cases assuming that the migration is performed through a communication channel with total bit rate R. A useful parameter is the ratio between the dirtying rate and the transmission channel bit rate

$$\lambda = \frac{P\,D}{R}.$$

A. Serial Migration of Multiple VMs

When the M VMs are migrated one at a time, each transfer is performed at full channel bit rate, i.e. $R_j = R,\ \forall j$. In this case, $\lambda_j = \lambda$ and each VM is migrated in $n_j = n^{(s)}$ rounds, according to (4). Therefore, the serial migration time of the whole VM set is given by

$$T_{\rm mig}^{(s)} = \sum_{j=1}^{M} T_{{\rm mig},j} = M\,\frac{V_{\rm m}}{R}\,\frac{1-\lambda^{n^{(s)}+1}}{1-\lambda} \quad (6)$$

The downtime of the whole VM set starts when the first VM is stopped at the source host (i.e., when the last iteration of the first VM begins) and ends when the last VM is resumed at the destination host. If $T_{\rm res}$ is the fixed time required to resume a VM at the destination host, the serial migration downtime can be computed as

$$T_{\rm down}^{(s)} = \frac{V_{\rm m}}{R}\,\lambda^{n^{(s)}} + (M-1)\,\frac{V_{\rm m}}{R}\,\frac{1-\lambda^{n^{(s)}+1}}{1-\lambda} + T_{\rm res} \quad (7)$$

B. Parallel Migration of Multiple VMs

When the M VMs are migrated simultaneously, each one of them is transferred at a bit rate that depends on how the bandwidth is shared among the on-going connections. Assuming an equal share of the channel bandwidth and considering that all the VMs in the set have the same memory profile, all VMs start and end their migration at the same instants. Therefore, there are always M simultaneous transfers and the transfer rate seen by each VM is $R_j = R/M,\ \forall j$. In this case, $\lambda_j = M\lambda$ and each VM is migrated in $n_j = n^{(p)}$ rounds, according to (4). The parallel migration time of the whole VM set is equivalent to the migration time of any single VM and is given by

$$T_{\rm mig}^{(p)} = T_{{\rm mig},j} = M\,\frac{V_{\rm m}}{R}\,\frac{1-(M\lambda)^{n^{(p)}+1}}{1-M\lambda} \quad (8)$$

The parallel migration downtime of the whole VM set corresponds to the last iteration (stop-and-copy phase) of any single VM and is given by

$$T_{\rm down}^{(p)} = M\,\frac{V_{\rm m}}{R}\,(M\lambda)^{n^{(p)}} + T_{\rm res} \quad (9)$$

C. Comparing Serial and Parallel Migration

From the formulas obtained above we can draw some conclusions about how the serial migration of a set of VMs compares to the case of parallel migration. If we compare equations (6) and (8) we can immediately see that this is not a trivial case of serial vs. parallel data transfer. In fact, migrating a sequence of M VMs at rate R does not require the same amount of time as migrating M VMs in parallel each at rate R/M, because the iterative live migration process and the fixed memory page dirtying rate produce different amounts of data to be transferred in the two cases.

In particular, we can prove that the parallel migration time is larger than the serial migration time. If we subtract (6) from (8) when $n^{(s)} = \log_\lambda V_{\rm th}/V_{\rm m}$ and $n^{(p)} = \log_{M\lambda} V_{\rm th}/V_{\rm m}$, we obtain

$$T_{\rm mig}^{(p)} - T_{\rm mig}^{(s)} = \frac{M\lambda\,(M-1)\,(V_{\rm m}-V_{\rm th})}{R\,(1-\lambda)\,(1-M\lambda)} \quad (10)$$

which is always non-negative under our assumptions. Similarly, we can prove that the serial migration downtime is larger than the parallel migration downtime, obtaining from (7) and (9)

$$T_{\rm down}^{(s)} - T_{\rm down}^{(p)} = \frac{(M-1)\,(V_{\rm m}-V_{\rm th})}{R\,(1-\lambda)} \quad (11)$$

which, again, is always non-negative.

These results suggest a possible trade-off: on the one hand, parallel migration is better than serial migration in terms of service provisioning, since the downtime is smaller; on the other hand, serial migration shows a smaller total transfer time and thus is better than parallel migration in terms of communication resource usage and transmission overhead.

IV. Numerical Results

In this section we present some numerical results obtained with the multiple VM live migration model introduced above. In order to represent a realistic scenario for the virtualized edge network architecture, we assign the following reference values to the model parameters:

- $M = 8$;
- $R = 1\,\text{Gbps}$, a reasonable bit rate available for inter-data center communication within the network infrastructure that provides connectivity to the virtualized edge networks we are considering;
- $V_{\rm m} = 1\,\text{GB}$, a typical value in current virtualization technologies and large enough to allow the execution of the required network functions;
- $P = 4\,\text{KB}$, the most common memory page size adopted by modern operating systems;
- $D = 2500\,\text{pps}$, corresponding to $\lambda = 0.082$; this is a reasonable estimation of the dirtying rate based on benchmark measurements [9] and considering that: (i) the VMs performing network functions are expected to be

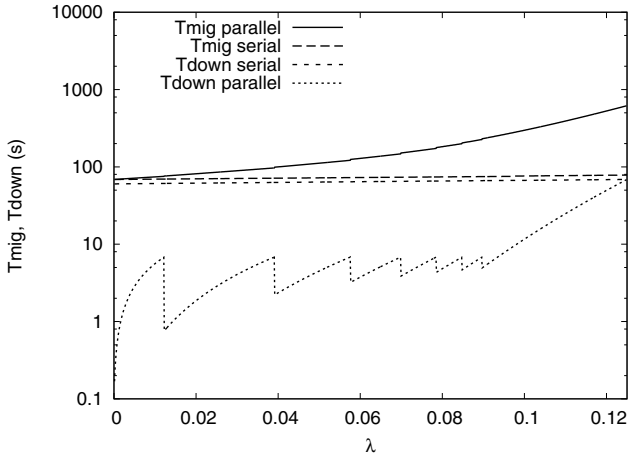

Fig. 1. Serial vs. parallel total migration time and downtime as a function of the ratio of dirtying rate over channel bit rate.

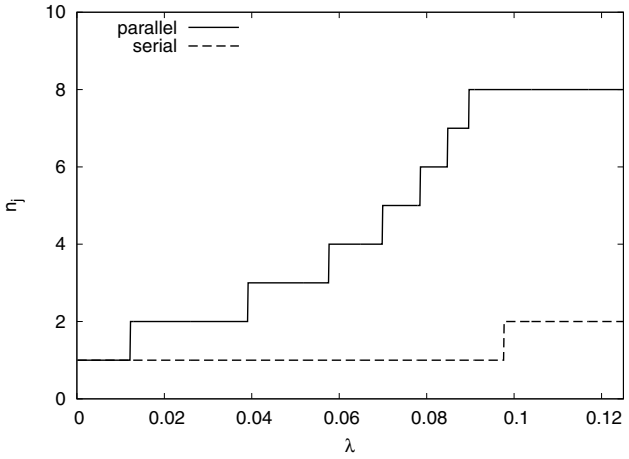

Fig. 2. Number of iterations required to migrate each VM as a function of the ratio of dirtying rate over channel bit rate.

not extremely demanding in terms of memory updates; (ii) the most frequently modified memory pages (the so-called *Writable Working Set*) are typically transferred only during the stop and copy phase, thus reducing the amount of dirty pages to be copied in each round [7];

- $V_{\mathrm{th}} = 100\,\mathrm{MB}$, a reasonable compromise considering the value of the other parameters;
- $n_{\max} = 8$, an estimated realistic value to limit the total transfer time of each VM;
- $T_{\mathrm{res}} = 100\,\mathrm{ms}$, a conservative estimation considering the current virtualization technologies.

The charts included in this section show the performance trends when one of the model parameters varies within its validity range, while all the other parameters are fixed to the reference values.

Figure 1 compares the serial and parallel migration strategies, in terms of both total migration time and downtime, as a function of the ratio of dirtying rate over channel bit rate. In this case the validity range $\lambda < 1/M$ is determined by the parallel migration. The figure allows to quantify the existing trade-off between serial and parallel migration already mentioned in subsection III-C. The logarithmic y-axes scale is used because the parallel migration downtime is extremely shorter than the serial migration downtime, while the total migration time is not too much larger as long as the dirtying rate is small. However, when λ goes beyond 0.04, $T_{\mathrm{mig}}^{(p)}$ grows significantly fast, whereas the serial migration shows much less sensitivity in this range.

This is mainly due to the different number of iterations needed to migrate each VM in the two cases. As shown in Fig. 2, the reduced transfer rate available to the single VM causes a quick increase in the number of iterations required in the parallel migration, whereas in the serial migration the full-rate transfer of each VM is completed in one or two iterations. The saw-tooth shape of the curves in Fig. 1 (quite visible for $T_{\mathrm{down}}^{(p)}$, but actually present in all the other curves) is caused

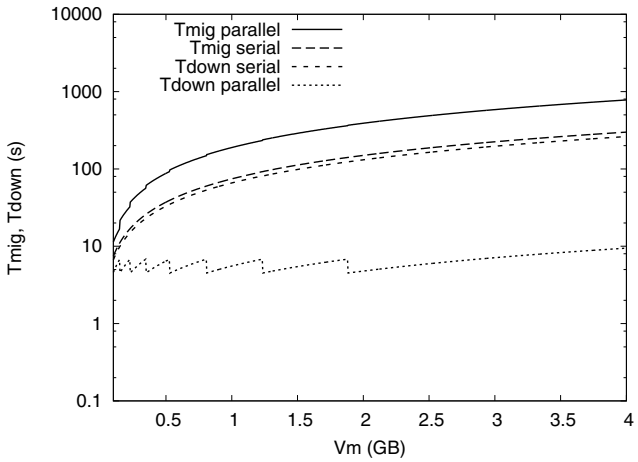

Fig. 3. Serial vs. parallel total migration time and downtime as a function of the VM memory size.

by the ceiling function present in (4), up to the point in which the number of iterations reaches $n_{\max}$, i.e., when $\lambda > 0.0897$ in the parallel migration case. Beyond that point, the total migration time grows more slowly with the dirtying rate, at the expenses of an increasing downtime.

Figure 3 compares the serial and parallel migration strategies as a function of the amount of memory allotted to each VM. In this case the validity range is $V_{\mathrm{m}} > V_{\mathrm{th}}$. Both serial and parallel total migration time as well as the serial downtime grow linearly with the VM memory size, as expected from (6), (7) and (8). The parallel downtime, instead, fluctuates around $5\,\mathrm{s}$ and starts to grow only when $n_{\max}$ is reached, i.e., when $V_{\mathrm{m}} > 1.87\,\mathrm{GB}$. This is due to the fact that, below that point, up to $V_{\mathrm{th}}\,\mathrm{MB}$ must be transferred during the stop & copy phase, thus keeping the downtime limited.

The role of the stop-and-copy phase threshold is displayed in Fig. 4. This parameter determines the required number of iterations as per (4). A large threshold reduces the value of n_j

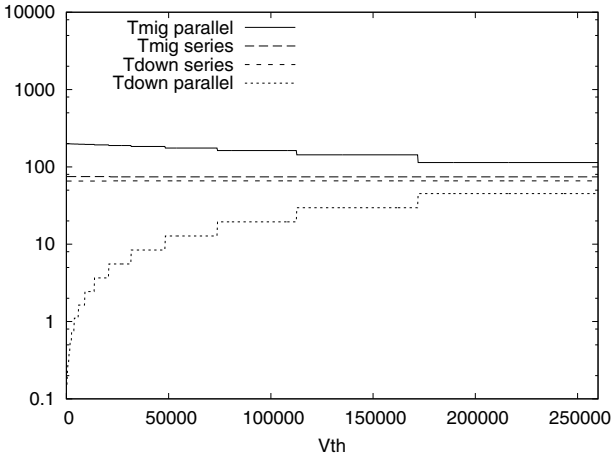

Fig. 4. Serial vs. parallel total migration time and downtime as a function of the stop-and-copy phase threshold.

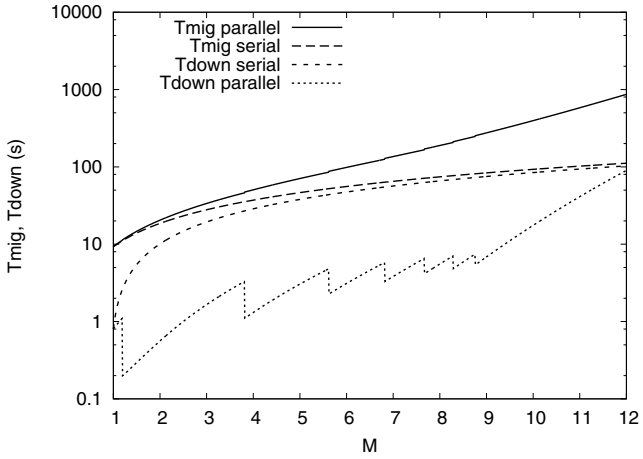

Fig. 6. Serial vs. parallel total migration time and downtime as a function of the number of VMs in the set.

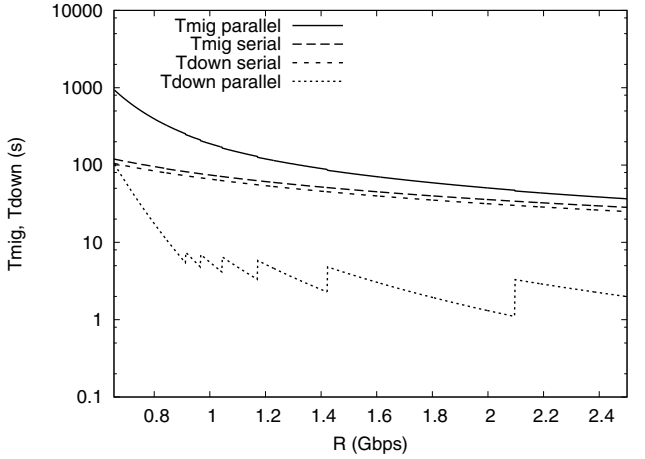

Fig. 5. Serial vs. parallel total migration time and downtime as a function of the transmission channel bit rate.

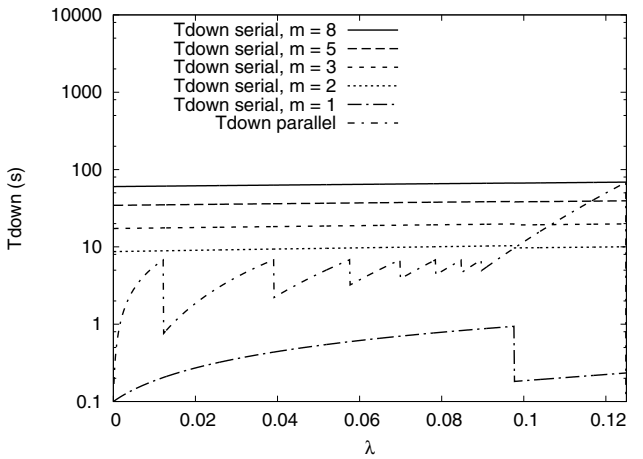

Fig. 7. Serial vs. parallel migration downtime as a function of the ratio of dirtying rate over channel bit rate for different sizes of the critical VM subset.

and then decreases the total migration time, but increases the downtime because more memory must be copied during the last iteration. With the reference values chosen, the parallel migration results much more sensitive to V_{th} than the serial migration due to the higher value of λ_j. The maximum number of iterations in the parallel migration is reached when $V_{\mathrm{th}} < 53.5\,\mathrm{MB}$.

The migration performance improves when the transmission channel bit rate increases, as shown in Fig. 5 where the validity range is $R > M\,P\,D$, as determined by the sustainability of the pre-copy algorithm in the parallel migration case. This figure is useful to dimension the inter-data center network bandwidth required to achieved a specified level of downtime and/or total migration time when the memory page dirtying rate is known. In this case, the parallel migration reaches the maximum number of iterations when $R < 913.71\,\mathrm{Mbps}$.

Figure 6 shows the impact of the size of the set of VMs to be migrated on the transfer performance. The parallel and serial migration strategies give the same result when $M = 1$, as obvious, and the difference becomes more significant when M increases, up to the validity limit of 12. As long as the number of VMs is kept limited, the parallel migration gives a very small downtime without exceeding too much with the total migration time. However, when M grows beyond 4, $T_{\mathrm{mig}}^{(p)}$ explodes, while $T_{\mathrm{mig}}^{(s)}$ keeps increasing linearly. The parallel migration reaches the maximum number of iterations when $M \geq 9$. When M approaches the validity limit the parallel migration downtime exponentially grows closer to the serial migration downtime, but with a much higher total migration time.

Finally, Fig. 7 shows how the serial migration downtime can be improved if we consider only a subset of the VMs as critical for the offered service. In fact, if we assume that only m out of M VMs are performing network functions that are strictly required to guarantee minimum connectivity service to the customer and if we give priority to these VMs in the migration process, we can redefine $T_{\mathrm{down}}^{(s)}$ as in (7) after

substituting M with m. The remaining $M - m$ VMs still to be migrated provide additional services that will be available to the customer with an additional delay. Figure 7 shows that the serial migration downtime could be very close to or even smaller than the parallel migration downtime if the critical VM subset is small, especially when the number of iterations in the parallel case has reached $n_{\max}$.

V. Conclusion

This manuscript presented a model to evaluate the performance of the live migration of a set of virtual machines cooperating to implement some sort of networking service. The service downtime due to the migration and the total migration time were derived for the two alternatives of serial and parallel migration strategies, showing that a trade-off exists. In particular, parallel migration provides a shorter downtime at the expense of a much longer migration time, meaning that it minimizes the service downtime at the cost of maximizing the communication resources occupancy and thus reducing their availability. The results presented also show that some interesting compromise could be achieved when just some of the virtual machines are strictly necessary to provide the basic connectivity service and therefore some of them may be temporarily switched off.

Acknowledgments

This work was partially supported by EINS, the Network of Excellence in Internet Science (http://www.internet-science.eu) through European Union's 7th Framework Programme under Communications Networks, Content and Technologies, Grant Agreement no. [288021].

References

[1] J. Sherry, et al., "Making Middleboxes Someone Else's Problem: Network Processing as a Cloud Service", Proceedings of the ACM SIGCOMM Conference, Helsinki, Finland, August 2012.

[2] M. Honda, et al., "Is it still possible to extend TCP?", Proceedings of the ACM SIGCOMM Conference on Internet Measurement (IMC), Berlin, Germany, November 2011.

[3] A. Manzalini, "Middle-Boxes? No thanks, Stateless Core and Stateful Edges", Telecom Italia Future Center, February 2013, available at http://www.blog.telecomfuturecentre.it/author/manzalini/

[4] The Open Networking Foundation, "Software-Defined Networking: The New Norm for Networks", ONF White Paper, April 2012, available at https://www.opennetworking.org/images/stories/downloads/white-papers/wp-sdn-newnorm.pdf

[5] "Network Functions Virtualisation", ETSI introductory white paper, October 2012, available at http://portal.etsi.org/NFV/NFV_White_Paper.pdf

[6] A. Manzalini, R. Minerva, F. Callegati, W. Cerroni, A. Campi, "Clouds of Virtual Machines in Edge Networks", IEEE Communications Magazine, Vol. 51, No. 7, pp. 63-70 , July 2013.

[7] C. Clark, et al., "Live Migration of Virtual Machines", Proceedings of the 2nd USENIX Symposium on Networked Systems Design & Implementation (NSDI), Boston, MA, May 2005.

[8] K. Z. Ibrahim, S. Hofmeyr, C. Iancu, E. Roman, "Optimized Pre-Copy Live Migration for Memory Intensive Applications", Proceedings of the International Conference of High Performance Computing, Networking, Storage and Analysis (SC), Seattle, WA, November 2011.

[9] H. Liu, H. Jin, C.-Z. Xu, X. Liao, "Performance and energy modeling for live migration of virtual machines", Cluster Computing, Springer, Vol. 16, No. 2, pp. 249-264, June 2013.