Improving Virtual Machines Networking Performance for Cloud Computing

Manel Bourguiba §, Ines El Korbi ‡, Kamel Haddadou †and Guy Pujolle †
§LRI, Paris-Sud University, Orsay, France
†LIP6, Pierre & Marie Curie University, Paris, France
‡Cristal Lab, ENSI, University of Manouba, Tunis, Tunisia
manel.bourguiba@lri.fr, kamel@gandi.net
guy.pujolle@lip6.fr, ines.korbi@ensi.rnu.tn

***Abstract*—Virtualization is a key technology to enable cloud computing. It enhances resource availability and offers high flexibility and cost-effectiveness. However, the driver domain based model for network I/O virtualization exhibits poor networking performance. To overcome this limitation, we proposed a packet aggregation based I/O model in a previous work. In this paper, we studied the impact of the aggregation on the packets delay. For this purpose, we proposed a dimensioning tool based on the queueing theory modeling of the aggregation based I/O virtualization. The proposed tool allowed us to dynamically tune the aggregation mechanism in order to achieve the best tradeoff between the packets delay and throughput. This allows the cloud infrastructure providers to meet the user applications requirements.**

I. Introduction

Cloud computing is a novel paradigm that provides infrastructure, platform, and software as a service. A cloud platform can be either virtualized or not. Virtualizing the cloud platform increases the resources availability and the flexibility of their management (allocation, migration, etc.). It also reduces the cost through hardware multiplexing and helps energy saving. Virtualization is then a key enabling technology of cloud computing. System virtualization refers to the software and hardware techniques that allow partitioning one physical machine into multiple virtual instances that run concurrently and share the underlying physical resources and devices.

Network intensive applications are among the applications dominating the cloud data-centers today [9]. In order to make the Cloud a more compelling paradigm, it is important to make the networking performance of the virtual machines scale up at line rates. Virtual machines typically share the access to the network device through a special virtual machine called driver domain. Network device sharing offers high flexibility and resource multiplexing. However, it drastically affects the virtual machines networking performance. We addressed this issue in [8] by proposing a packet aggregation based mechanism that transfers packets in containers between the driver domain and the virtual machines. The proposed mechanism allows the reception, transmission and forwarding performance of virtual routers to scale up at line rates. However, it introduces an additional delay to the packets latency.

In this paper, we study its impact on packets delay and jitter. Then we propose a dimensioning tool that allows to dynamically tune the packet aggregation mechanism based on the required flows QoS, the system load and the number of concurrent virtual machines sharing the same network device. An adequate tuning enables the system to achieve the best tradeoff between throughput and delay during the packet aggregation. For this purpose, we modeled the proposed aggregation mechanism using queuing theory. The proposed analytical model enabled us to determine the total transmission delay of a packet in the aggregation-based system with regard to the service time values provided by the experimental evaluation of the system.

The remainder of this paper is organized as follows. In section II, we give an overview of the background of our work and state the problem. In section III, we introduce the dimensioning tool that dynamically tunes the aggregation mechanism. Section IV describes some related works. Finally, section V concludes the paper and introduces our future work.

II. Background and Problem Statement

A. Native Network I/O virtualization

Virtualization relies on the virtual machines monitor (VMM) or hypervisor to enable multiple virtual machines to share the same physical machine while ensuring software isolation among them. Access to the devices and namely to the network device is shared between all the virtual machines through a special virtual machine called driver domain. The driver domain hosts the real device drivers and has a direct and exclusive access to the devices. The received packet has to first pass through the hypervisor which relays it to the driver domain. Then the driver domain transfers it to the corresponding virtual machine through the shared memory. This I/O networking model performance is limited due to the overhead incurred by the communication between the driver domain and the guests. We addressed this issue by proposing an aggregation based I/O mechanism [8].

B. Packet aggregation based I/O virtualization

The I/O communication between the driver domain and the virtual routers requires multiple costly memory transactions [8]. It would be judicious to use train queueing techniques to aggregate multiple packets into a group of packets that will

978-3-901882-50-0

be transferred as a unit. Below, we present a summary of the aggregation based mechanism proposed in [8]. It is composed of two parts: a Container and an Unloader, laying one in the driver domain and the other in the virtual machine. Upon the arrival of a packet, the container generator module removes its MAC header and checks whether a container with that same MAC destination is already waiting to be transferred to the netfront. If so, the container generator checks whether the total size of the container would exceed the maximum allowed size of the container after adding the incoming packet. If so, the container is transferred to the destination through the shared memory and the generator creates a new container with the same MAC header of that incoming packet. If not, it appends an offset of two bytes to the container in which it stores the packet's size useful to determine the position of the next packet in the container. Then the packet itself is queued at the tail of the container. The packets in containers remain queueing until the container's maximum size is reached or its timeout expires. After that, the container is transferred through the shared memory to the VM. In the case where no container is available to carry the packet, the container generator creates a new container with the same MAC header of the arriving packet. This container first contains the packet's offset followed by the packet itself. Each container is associated a timeout to prevent packets from suffering from unacceptable delays. The timeout is then set to its initial value at the moment of generation of the container. Each container is also associated a maximum size. The Unloader is the component that unloads the received container at the destination in order to extract the packets. It removes its MAC header and retrieves the packets one by one based on their offset. Packets are then processed by the upper layer. Packets transferred from the virtual machine to the driver domain follow the opposite path. They are enqueued in containers that will be then unloaded at the driver domain as described before.

C. Impact on Delay and Jitter

Since packets queuing techniques introduce an additional delay to the traffic, we need to investigate the delay performance. Studying the delay performance is important since some network applications do not tolerate exceeding a threshold latency.

1) Experimental setup: The system under test is a Dell PowerEdge 2950 server, with two 2940 Mhz Intel Quad-core CPUs. Pairs of cores share the same L2 cache, and all eight cores share the same main DDR2 667Mhz memory arranged in eight 1Gb modules. Networking is handled by one quad-gigabit card using a PCI X4 channel. The e1000 driver is used with NAPI enabled. As a hypervisor, we use Xen 3.4.0 in para-virtualization mode [4] and with the aggergation mechanism installed on the driver domain and the guest machine. We instantiate one virtual machine as a driver domain and one guest machine for traffic generation and forwarding. The virtual machines run an IA-64 Linux kernel in the 2.6.24.7 version. As traffic sink and source, we used two NEC machines with 2400 Mhz core 2 duo processors, 1GB DDR2 667Mhz and 1 Gigabit NIC each. Each machine runs an IA-64 Linux Kernel in the 2.6.24.7 version. We used Click [11] for the traffic forwarding. We generated a UDP flow with 64 bytes sized packets. 64 bytes represents the smallest data packets size and it has historically been the reference benchmark used by network equipment vendors. We set the container size to 4096 bytes and its timeout to 200 μs. We measured the packet's forwarding delay by computing the difference between the instant at which the packet arrives to the network device and the instant at which it leaves the system under test. The jitter is computed following RFC 1889 [19]. We plotted the maximum jitter since it better shows the impact of the aggregation on the variation of the delay. Typically the difference between the delays of the last packet of a container and the first packet from the next container is very high compared to the difference between the delays of two adjacent packets in the same container.

2) Experimental results: Figure 1 shows the packets mean delay and jitter variation as a function of the input rate with the native system for one virtual machine forwarding traffic. We notice that with the native system, the maximum jitter drastically increases at the input rate of 80Kp/s which represents the maximum achievable forwarding throughput. However, the delay remains almost static around 120 μs since it is computed for the packets that have not been rejected during their forwarding only. With the aggregation based system, we notice the same behavior for the maximum jitter when the system reaches the maximum achievable throughput of 800 Kp/s (Figure 2). Please refer to [8] for details about the throughput improvement with the aggregation based system. The delay behaves differently. We first notice the obvious increase of the packets average delay due to the additional waiting time until the container timeout expires or the maximum size is reached. This delay is of 280μs for an input rate of 100 Kp/s. At this rate, the delay was limited to 50μs with the native system.

Then, note that the delay starts decreasing from an input rate of 600 Kp/s. In fact, at this high input rate, the containers reach their maximum size rapidly before the timeout has expired which triggers the container transfer. At low input rates (up to 200 Kp/s), containers are transferred to destination after the expiration of the timeout, fixed to 200μs. In fact, at low input rates, containers spend a long time to get full. In this case, packets mean waiting time is 200μs for packets forwarding (100μs in the driver domain and 100μs in the VM). For input rates beyond 200 Kp/s, containers are transferred as soon as their maximum size has been reached. Packets delay in the system then reaches 400μs. This delay will decrease for higher input rates (starting from 600 Kp/s). At this rate, containers loading is fast enough to allow small waiting times for the packets. The jitter remains acceptable until an input rate of 900 Kp/s.

Obviously, better throughput is achieved with larger container (hence bigger timeouts) which results in longer delays. Hence, in order to make the aggregation a totally advantageous mechanism, we need an efficient dynamic aggregation tuning

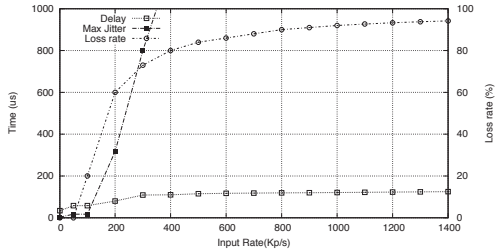

Fig. 1. Packets delay, jitter and loss rate with the native system

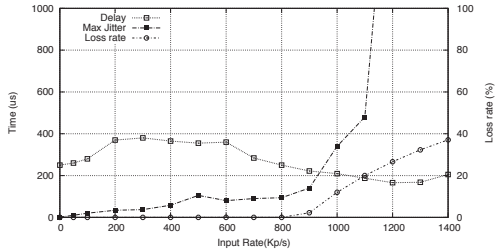

Fig. 2. Packets delay, jitter and loss rate with the aggregation based system

to guarantee to real time packets an acceptable delay and throughput. Next we propose a tool that dynamically adjusts the container size according to the application requirements on delay while offering the best tradeoff with the throughput.

III. Packet aggregation dimensioning tool

Next, we will use the queuing theory to propose an analytical model for our system composed of the driver domain, the virtual machines and the shared memory. Then we will resolve the proposed model in order to get the average waiting time of the packets and therefore the average packets forwarding delay. Analytically modeling the system offers a means to tune it in a generic fashion with regard to its variable characteristics (load, number of concurrent virtual machines, etc.).

We consider a system composed of a driver domain and N virtual machines that we denote by VM_i, i=1..N that exchange containers through the shared memory. To model our system, we consider the following assumptions:

Assumption 1: Since we are interested in evaluating the maximum achievable throughput, we assume the system to be under heavy loaded conditions. In such conditions, there will always be a container available to be unloaded in the guest domain.

Assumption2: The packets arrival to the driver domain (coming from the network device) follows a Poisson process with parameter λ_0. At network entries, the real-time flows arrive according to a Poisson process. The arrival process of the Internet traffic, at a packet scale, does not correspond to a Poisson process (On/off sources for example). However, it can be modeled by a Poisson process [16].

Upon the arrival of a container to the virtual machine, this latter extracts the packets from the container and sends them to the upper layers to be processed. The packets are then sent back to the netfront randomly to get aggregated again in containers which will be transferred further to the netback. This leads to the assumption 3 as follows:

Assumption 3: In the virtual machine i, i=1..N, packets arrive from upper layers to the netfront to get aggregated according to a Poisson process with parameter λ'_i, i=1..N.

Assumption 4: Containers receive an exponential service of parameter μ_0 when they are transferred from the driver domain to the virtual machines and of parameter μ_i, i=1..N when they are transferred from the virtual machine VM_i, i=1..N to the driver domain. This assumption simplifies the system modeling and generally results in a majoring of the real system behavior. Indeed, it is possible to major an M/D/1 queue (with a constant service) by an M/M/1 queue where the service time is exponential [14].

Assumption 5: We also assume that packets extraction time is null, in the driver domain as well as in the virtual machines.

Assumption 6: We further assume that the containers generated by both the driver domain and the virtual machines have the same size k.

A. Proposed analytical model

Upon the arrival of a packet from the network device to the driver domain, this latter creates a container with size k_0 and waits for the arrival of the k_0-1 next packets before transferring the whole container through the shared memory to the destined virtual machine. Then the delay between the arrival of two containers generated by the driver domain and transferred to the shared memory corresponds to the sum of k_0 random variables. Each random variable is exponentially distributed with parameter λ_0 (cf. assumption 2). Kleinrock proves in [15] that the sum of k_0 exponential random variables is characterized by an Erlang distribution with parameters λ_0 and k_0.

In the same way, within each virtual machine i, i=1..N, upon the arrival of a packet from the upper layer, the virtual machine creates a container with size k_i, i=1..N and waits for the arrival of k_i-1 next packets before transferring the container to the driver domain through the shared memory. Then, the time that separates the arrivals of two successive containers generated by the virtual machine to the shared memory corresponds to the sum of k_i, i=1..N random variables. Each one of those variables follows an exponential process with parameter λ_i, i=1..N (assumption 3).

Similarly, according to [15], the containers will arrive to the shared memory according to an Erlang distribution with parameters λ_i and k_i, i=1..N. We can then conclude that the containers arrival process to the shared memory is the result of the superposition of N+1 Erlangian processes Er(λ_i,k_i), i=0..N. The Erlangian process Er(λ_0,k_0) correspond to the arrival of the containers from the driver domain to the shared memory while the Erlangian processes Er(λ_i,k_i), i=1..N corresponds to the arrival of the containers from the virtual machine VM_i, i=1..N to the shared memory. In the shared memory, the containers will get service from a single FIFO server according to an exponential process with parameter μ.

The arrival process of packets to the server which is a superposition of multiple Erlangian processes is not a clas-

sical process with known characteristics. Next, we propose a characterization of the superposition of multiple Erlangian processes.

1) Characterization of the packets arrival process to the shared memory: According to [16], the Erlang distribution is a particular case of the class of continuous distributions called continuous Phase Type distributions and denoted by PH distribution. A random variable associated with a continuous PH distribution of order γ represents the time to absorption in a γ -state Markov chain and has the cumulative distribution function: 1-αexp(Tt)***1*** , where the γ-dimensional row vector α is the initial distribution of the Markov chain, T is its infinitesimal generator matrix, and ***1*** is a γ-dimensional column vector of ones.

Each Erlang process Er(λ_i,k_i), i=0..N that models the arrival of packets is then characterized by the pair (α_i,T_i), i=0..N where α_i is a row vector of dimensions k_i and T_i the generator matrix of dimensions $k_i * k_i$, i=0..N.

$$\alpha_i = \begin{pmatrix} 1 & 0 & ... & 0 \end{pmatrix}, \text{ i=0..N}$$

$$\text{and } T_i = \begin{pmatrix} -\lambda_i & \lambda_i & & & \\ & -\lambda_i & \lambda_i & & \\ & & -\lambda_i & ... & \\ & & & ... & \lambda_i \\ & & & & -\lambda_i \end{pmatrix}, \text{ i= 0..N}$$

In [17], the authors have shown that the superposition of N PH processes with parameters (α_i, T_i), i=1..N also remains phase-type distributed and is characterized by: $[\alpha, T]$ where $\alpha = [p(1)\alpha(1) \otimes p(2)\alpha(2) \otimes ... \otimes p(N)\alpha(N)]$

$$\text{and T= } \begin{pmatrix} T(1) & 0 & ... & 0 \\ 0 & T(2) & ... & 0 \\ ... & ... & & ... \\ 0 & 0 & ... & T(N) \end{pmatrix}$$

Where $[\alpha(1), T(1)] = [\alpha_1 \otimes \alpha_2 \otimes ... \otimes \alpha_n, T_1 \otimes I_2 \otimes ... \otimes I_N + I_1 \otimes T_2^* \otimes I_3 \otimes ... \otimes I_N + ... + I_1 \otimes I_2 \otimes ... \otimes I_{N-1} \otimes T_N^*]$ and $[\alpha(2), T(2)] = [\alpha_1 \otimes \alpha_2 \otimes ... \otimes \alpha_n, T_1^* \otimes I_2 \otimes ... \otimes I_N + I_1 \otimes T_2 \otimes I_3 \otimes ... \otimes I_N + ... + I_1 \otimes I_2 \otimes ... \otimes I_{N-1} \otimes T_N^*]$ and so on until $[\alpha(N), T(N)] = [\alpha_1 \otimes \alpha_2 \otimes ... \otimes \alpha_n, T_1^* \otimes I_2 \otimes ... \otimes I_N + I_1 \otimes T_2^* \otimes I_3 \otimes ... \otimes I_N + ... + I_1 \otimes I_2 \otimes ... \otimes I_{N-1} \otimes T_N]$.

$\otimes$ denotes the Kronecker product [16]. I_i is the identity vector and (α_i, T_i^*) is the distribution of the residual time of the Erlang(λ_i,k_i) process which is a Mixture of Generalized Erlangs (MGE) distribution. MGE distributions are also proven to be PH. The expression of (α_i, T_i^*) is given in [16] as follows:

$\alpha_i^T = (1, 0, ..., 0)$ and T_i^* =

$$\begin{pmatrix} -\lambda_i & ((k_i-1)/k)\lambda_i & & & \\ & -\lambda_i & ((k-2)/(k-1))\lambda_i & & \\ & & -\lambda_i & ... & \\ & & & ... & ... \\ & & & ... & 1/2\lambda_i \\ & & & & -\lambda_i \end{pmatrix}$$

Then, the superposition of the Erlangian processes that model the containers arrival to the shared memory and which is characterized by (α_i, T_i), i=0..N is a phase type process characterized by $[\alpha, T]$ as defined above.

2) Characterization of the service process: We have assumed that the service time of a container follows an exponential process with parameter μ. The exponential law is a particular case of a phase type process with a phase equal to 1.

We can therefore model our system consisting of the driver domain, the virtual machines and the shared memory with a queue station of type PH/PH/1. This queue consists of one FIFO server with a PH arrival process and a PH service time. This type of queue station does not correspond to neither a birth and death process nor a matrix-base queue station. To resolve such a queue station and evaluate its performance, we will use a methodology based on matrix analysis and Quasi Birth and Death processes denoted by QBD. Next, we explicit how we will use the QBD processes in the resolution of the PH/PH/1 queue station used to model our system.

B. Modeling a PH/PH/1 queue station by a QBD process

Let's consider a queue station of type PH/PH/1 where the inter-arrivals and the service time are phase type processes. Inter-arrivals are of parameters (τ,T) and of order m_T and the service times are of parameters (α, S) and of order m_S. τ and α represent the initial probability vectors while T and S represent the transition generators for the transient states of the arrival process and the service process respectively.

The station PH/PH/1 can be modeled by the Markovian process {(N(t), a(t), s(t): t≥0)} in the state space {(n,a,s): n≥0, 1≤ a ≤ m_T, 1≤ s ≤ m_S} where:

- N(t) denotes the number of customers in the system at instant t.
- a(t) denotes the phase of the arrival process.
- s(t) denotes the phase of the service process.

Given that the transitions occur individually, they are restricted to adjacent levels. The markovian process that models our queue station is then a continuous QBD and its infinitesimal generator Q is given by: $\begin{pmatrix} B_1^* & A_0^* & & \\ A_2^* & A_1^* & A_0^* & \\ & A_2^* & A_1^* & ... \\ & & ... & ... \end{pmatrix}$

where A_0^*, A_1^*, A_2^* and B_1^* are of orders m_S and m_T and are given by:

$$A_0^* = t.\tau \otimes I \quad (1)$$

$$A_1^* = T \otimes I + I \otimes S \quad (2)$$

$$A_2^* = I \otimes s.\sigma \quad (3)$$

$$B_1^* = T \otimes I \quad (4)$$

with t = -T.***1*** and s= -S.***1*** where ***1*** is a column vector with all components equal to 1 and $\otimes$ denotes the Kronecker product [16].

Furthermore, it is known that the stationary distribution of such a process is matrix-geometric [18]. The stationary row vector π^* is then decomposed as follows:

π^*= (π_0^*, π_1^*,...,π_n^*) where π_i^* is the sub-vector of order i. There exists a matrix R^* of order $m_S m_T$ as:
π_n^* = π_0^* $(R^*)^n$, for n$\geq$ 1.

R^* is the minimal nonnegative solution of the nonlinear matrix equation :

$$A_0^* + R^* A_1^* + R^{*2} A_2^* = 0 \quad (5)$$

Furthermore, the stationary probability vector π_0^* is uniquely determined by solving the boundary condition:

$$\pi_0^* B_1^* + \pi_1^* A_2^* = \pi_0 (B_1^* + R^* A_2^*) = 0 \quad (6)$$

and the normalization condition:

$$\sum_{i=0}^{\infty} \pi_i^* e = \pi_0^* (I - R^*)^{-1} e = 1 \quad (7)$$

Where I denotes the identity matrix of appropriate dimension.

This theorem is due to Neuts for the case $M < \infty$ [2]; the theorem for the case M=∞ follows the results of Tweedie [5]. It is known that the stationary distribution is determined once R^* is obtained. Several iterative algorithms exist for numerically solving (5). An overview of such algorithms is provided in [10]. A related matrix, typically denoted by G, also plays an important role together with R^* in the genaral theory of matrix-analytic methods. This related G matrix is the minimal nonnegative solution of the nonlinear matrix equation:

$$A_0^* G^2 + A_1^* G + A_2^* = 0 \quad (8)$$

(see [18]). Some recent algorithms for numerically solving (5) involve first computing G and then computing the matrix R^* based on the relationship:

$$R^* = A_0^* (-(A_1^* + A_0^* G))^{-1} \quad (9)$$

First we calculate R^* following the procedure described in [18]. The evaluation of R^* allows us then to calculate the average waiting time of the clients in the system.

C. Average waiting time in the queue station PH/PH/1

The packets waiting time process in a PH/PH/1 queue station has been proven to be also PH in [18]. It can be characterized by the pair $(wait_\alpha, wait_T)$. In order to determine $(wait_\alpha, wait_T)$, the authors of [18] defined:

$$\tilde{\sigma} = (\alpha S^{-1} \boldsymbol{1})^{-1} \alpha S^{-1} \quad (10)$$

$$\Delta = diag(\tilde{\sigma}) \quad (11)$$

We then have:

$$wait_T = \Delta^{-1} (S + R^* S \alpha)^{-1} \Delta \quad (12)$$

Furthermore, they defined:

$$\rho = (\beta S^{-1} \mathbf{1}) / \alpha T^{-1} \mathbf{1} \quad (13)$$

$$\sigma_\rho = \rho \tilde{\sigma} \quad (14)$$

$$\theta = (-\alpha S^{-1} \mathbf{1}) t(\mathbf{s}) \Delta \quad (15)$$

$$D = \Delta^{-1} t(R^*) \Delta \quad (16)$$

where t(M) is the transpose of matrix M and diag(M) the diagonal of matrix M. ρ is the system load such as $\rho < 1$. $wait_\alpha$ is then given by:

$$wait_\alpha = (1 - (\alpha(\mathbf{1} - R^*)^{-1})\mathbf{1}) \alpha \mathbf{1} (\theta D \mathbf{1})^{-1} \theta D \quad (17)$$

The average waiting time is then given by:

$$E(Waiting\_time) = -t(wait_\alpha (wait_T)^{-1}) \mathbf{1} \quad (18)$$

It is henceforth possible to determine the probability with which the waiting time exceeds the value y, based on the size of the containers as follows:

$$P[Waiting\_time > y] = wait_\alpha e^{wait_T y} \mathbf{1} \quad (19)$$

D. Analytical model validation

We have implemented and solved the proposed model using the Matlab environment. First, we validate the proposed model through a comparison of the analytical and experimental results with regard to the packet average sejourn time in the system. Then, we present a use case of the proposed tool. Evaluating the packets waiting time in the system allows us to evaluate the total forwarding delay of the packets in the system. This latter corresponds to the time needed to transfer the packet from the driver domain to the virtual machine and then from the virtual machine to the driver domain, to which we add the waiting time experienced by the packet when waiting for the containers to get full before each transfer. Within the virtual machine, the packets do not receive any specific processing since they are directly forwarded to the output interface. The time spent by the packets in the virtual machine is then quasi null and assumed to be null in our model. In order to validate the proposed model, we measured the packets average forwarding delay with four virtual machines. The containers size is set to 200 packets, and their timeout to 200μs. In order to obtain analytical results concerning the average sejourn time of the packets, we injected to the proposed model the service rate of the containers obtained through experimentations. It corresponds to the inverse of the service time of the containers, which corresponds to the time needed to transfer the container from the netback to the netfront and vice-versa through the memory. Figures 3 shows the average sejourn time of the packets as a function of the input rate. They show both the analytical and experimental results. Note that for an input rate up to 600 Kp/s, the experimental average sejourn time of the packets is well below their analytical sejourn time. This is due to the fact that for low input rates, the containers are transferred upon the expiration of their timeout and not upon reaching the maximum size. Since we set the timeout to 200 μs, the mean delay remains below 400μs. At high input rates, we note that the analytical sejourn time coincides with the experimental one. This allows us to conclude that the proposed model reproduces in a convenient way the modeled system.

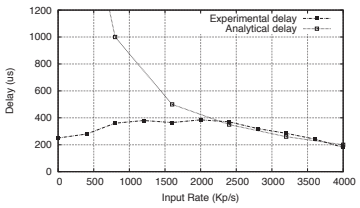

Fig. 3. Experimental and analytical waiting time with four virtual machines

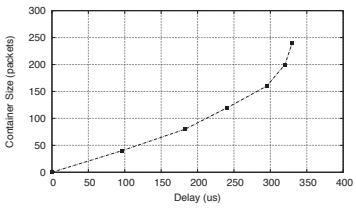

Fig. 4. Container size as a function of the required delay

E. Analytical results: Container size based on the required delay

Figure 4 shows the container size that corresponds to the forwarding delay that has been required by the application. It corresponds to the sum of the packets waiting time and their service time. The forwarding throughput is of 800 Kp/s. We will use this time to evaluate the total forwarding delay of the packet. Let us consider the case of an application requiring a maximum tolerable forwarding delay of 600μs. This delay corresponds to a one way delay of 300 μs. Figure 4 shows that in order to guarantee the required delay, we need a container size of 160 packets. Then packets belonging to these flows will then be aggregated in containers that will be transferred once their size reaches 160 packets.

IV. Related work

A fair number of research efforts has been dedicated to the improvement of the I/O performance through the enhancement of the communication scheme between the driver domain and the virtual machines. The grant reuse [7] enables the guest domain to greatly reduce the number of grant issue and revoke operations that are needed for I/O by taking advantage of temporal and/or spatial locality in its utilization of I/O buffers. The guest domain can issue a grant for a page containing I/O buffers, then use the page several times for I/O, and revoke access to that page. In contrast, in the existing implementation every I/O involves grant issue and revoke operations. Consequently, the grant reuse scheme reduces the number of grant hypercalls and page mapping/unmapping operations needed for I/O. In [12], the authors proposed a new design for the memory sharing mechanism with Xen. The basic idea of the new mechanism is to enable the guest domains to unilaterally issue and revoke a grant. This allows the guest domains to protect their memory from incorrect Direct Memory Access (DMA) operations. Large Receive Offload (LRO) [20] combines multiple received TCP packets and passes them as a single larger packet to the upper layer in the network. By reducing the number of packet processing operations, the CPU overhead is lowered and a performance improvement is expected. LRO allows an improvement from 1000 Mb/s to 1900 Mb/s with five 1Gb/s Intel NICs. LRO is an aggregation mechanism at the receive side only, performed in the device driver, in the context of TCP packets reception. In the case of a virtualized system, LRO can be used to transfer packets from the NIC to the driver domain. However, the grant reuse as well as our proposal are schemes operating between the driver domain and the guest domains. XenSockets [21] is a one way communication pipe between two VMs. It implements an inter VM communication mechanism based on the Unix socket principle. The implementation of XenSockets is based on statically shared memory buffers. It omits the traditional Xen page flipping mechanism and uses shared memory for message passing instead. The sender creates a memory page and then remaps that page into the receiver's accessible address space. However Xensocket does not support VM migration and exhibits poor transparency.

V. Conclusion

In order to overcome the performance limitation of the native driver domain based I/O, we formerly proposed a new mechanism inspired from packet aggregation techniques that allows to transfer containers of packets at once. Experimental evaluation showed that the proposed packet aggregation mechanism significantly improves networking performance. In this paper, we studied the impact of aggregating packets on the delay. Given the additional waiting delay experienced by the packets during the aggregation operation, we proposed a new dimensioning tool to dynamically tune the aggregation mechanism in order to achieve the best tradeoff between the packets transmission throughput and the delay they experience in the system. The proposed tool is based on an analytical queueing theory modelling of the system composed of the virtual machines, the driver domain and the shared memory. Comparing the experimental and analytical results validates the model since it shows that it reproduces in a convenient way the system behavior. The model implementation allowed us to determine the average waiting time as a function of the container size, which offers a means to guarantee a packets' delay that respects the application requirements. As a future work, we will extend the model in order to dynamically dimension the physical machine by determining the number of virtual machines it can support given the system load, the forwarded flows requirements and the available physical resources. Furthermore, we will combine the proposed aggregation mechanism with recent hardware advances in multi-queue 10 Gbps network devices to make our platform more viable for virtualized cloud platforms. We believe such software enhancements combined with recent advances in hardware will enable better deployment of cloud platforms.

References

[1] J. Shafer, I/O Virtualization Bottlenecks in Cloud Computing Today, Workshop on I/O Virtualization (WIOV 2010), Pittsburgh, 2010.

[2] Neuts, M.F. (1981). Matrix-geometric Solutions in Stochastic Models, An Algorithmic Approach (The Johns Hopkins Press, Baltimore).

[3] G. Liao, D. Guo, L. Bhuyan, and S.R King, Software techniques to improve Virtualized I/O performance on multi-core systems, ANCS 2008.
[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R.Neugebauer, I.Pratt, and A. Warfield, Xen and the art of virtualization, 19th ACM Symposium on Operating Systems Principles, October 2003.
[5] Tweedie, R.L. (1982). Operator-geometric stationary distributions for Markov chains, with applications to queueing models (Advances in Applied Probability, vol. 14, pp. 368-391).
[6] P. Xing, L. Ling, M. Yiduo, A. Menon, S. Rixner, A.L Cox, W. Zwaenepoel, Performance Measurements and Analysis of Network I/O applications in Virtualized Cloud, International Conference on Cloud Computing, 2010.
[7] K.K Ram, J.R. Santos, Y. Turner, A.L Cox, and S. Rixner, Achieving 10Gb/s using safe and transparent Network Interface Virtualization, In VEE 2009.
[8] M. Bourguiba, K. Haddadou, G. Pujolle, Packet Aggregation Based Network I/O Virtualization for Cloud Computing, Elsevier Computer Communications, Volume 35, Issue 3, February 2012, pp 309-319
[9] A. Li, X. Yang, S. Kandula, M. Zhang, CloudCmp: comparing public cloud providers, IMC 2010
[10] Latouche, G., Ramaswami, V. (1999). Introduction to Matrix Analytic Methods in Stochastic Modeling (SIAM, Philadelphia).
[11] E. Kohler, R. Morris, B. Chen, J. Jahnotti, and M. F. Kasshoek, The click modular router, ACM Transactions on Computer Systems, vol. 18, no. 3, pp. 263-297, 2000.
[12] K.K Ram, Y. Turner and J.R, Santos, Redesigning Xen's memory sharing mechanism for safe and efficient I/O virtualization , In the second workshop on I/O virtualization, 2010.
[13] M. Hasan Jamal, A. Qadeer, W. Mahmood, A. Waheed, J.J. Ding, Virtual Machine Scalability on Multi-Core Processors Based Servers for Cloud Computing Workloads, International Conference on Networking, Architecture and Storage, 2009.
[14] "T. Bonald, A. Proutire, J. Roberts, "Statistical performance guarantees for streaming flows using expedited forwarding", IEEE INFOCOM2001,March 2001.
[15] L.Kleinrock, "Queuing Systems, Volume I: Theory", Wiley-Interscience, pp:137-138, 1975
[16] M. Neuts, Matrix-geometric solutions in stochastic models. an algorithmic approach. Queueing Syst. Theory Appl., 1981.
[17] G. Latouche and V. Ramaswami, A logarithmic reduction algorithm for quasi-birthand- death processes, Journal of Applied Probability, vol. 36, pp. 650674, 1993.
[18] V. Ramaswami, From the matrix-geometric to the matrix-exponential, Queueing Syst. Theory Appl., vol. 6, pp. 229260, June 1990.
[19] http://www.ietf.org/rfc/rfc1889.txt
[20] A. Menon and W. Zwaenepoel "Optimizing TCP Receive Performance", USENIX 08: 2008 USENIX Annual Technical Conference
[21] X. Zhang, S. McIntosh, P. Rohatgi, and J. L. Griffin, "XenSocket: A High-Throughput Interdomain Transport for Virtual Machines", Proc. ACM/IFIP/USENIX Middleware Conference (Middelaware' 07), 2007.