Evaluating, Estimating, and Improving Network Performance in Container-based Clouds

Cassiano Rista*, Marcelo Teixeira†, Dalvan Griebler*‡ and Luiz Gustavo Fernandes*

*Pontifical Catholic University of Rio Grande do Sul (PUCRS),
GMAP Research Group (FACIN/PPGCC), 6681, Ipiranga Av. – Porto Alegre – RS – Brazil
†Federal University of Technology - Paraná (UTFPR), Pato Branco – PR – Brazil
‡Três de Maio Faculty (SETREM), 2405, Santa Rosa Av. – Três de Maio – RS – Brazil,
Laboratory of Advanced Research on Cloud Computing (LARCC)
Email: luis.rista@acad.pucrs.br

***Abstract*—Cloud computing has recently attracted a great deal of interest from both industry and academia, emerging as an important paradigm to improve resource utilization, efficiency, flexibility, and pay-per-use. However, cloud platforms inherently include a virtualization layer that imposes performance degradation on network-intensive applications. Thus, it is crucial to anticipate possible performance degradation to resolve system bottlenecks. This paper uses the Petri Nets approach to create different models for evaluating, estimating, and improving network performance in container-based cloud environments. Based on model estimations, we assessed the network bandwidth utilization of the system under different setups. Then, by identifying possible bottlenecks, we show how the system could be modified to improve performance. We then tested how the model would behave through real-world experiments. When the model indicates probable bandwidth saturation, we propose a link aggregation approach to increase bandwidth, using lightweight virtualization to reduce virtualization overhead. Results reveal that our model anticipates the structural and behavioral characteristics of the network in the cloud environment. Therefore, it systematically improves network efficiency, which saves effort, time, and money.**

***Index Terms*—Cloud Computing; Network Performance; Formal Modeling; Petri Nets; Simulation.**

I. Introduction

Cloud computing has become a common computational resources in many organizations, universities, and research centers. Currently, cloud computing delivers infrastructure (IaaS), platform (PaaS), and software (SaaS) as services such as the pay-as-you-go model to customers [1]. These modern environments are responsible for executing data and computation intensive applications, often characterized by large amounts of transactions and workflows. Providing high throughput for these applications and avoiding network performance degradation is therefore of paramount importance.

Traditional cloud environments, which use virtualization technologies that are not optimized for the execution of network-intensive applications, add significant overheads [2]. Moreover, traditional networks are not designed for cloud computing and pay-per-use cloud billing. This means that customers need to monitor resource usage to stay within their budget. However, unpredictable networking costs can make this a difficult task.

In addition, modeling and estimating network performance of container-based cloud environments is an extremely complicated, because they are intrinsically complex both in terms of structure and number of components. Yet, the capability to predict the behavior of these systems is quite important for capacity planning and management of these environments.

Finally, it is important to design a correct evolution roadmap of each container-based cloud environment as well as keep potential scalability problems under control. This can be done by anticipating the effects of the deployment of new networking mechanisms, being able to explore the scalability limits of implemented applications, and by comparing different possible proposed strategies. Hence, these three aspects require the modeling language to be powerfully expressive and scalable, to effectively model the basic structural elements of the real cloud environment.

In the context of the Computers and Communications, an important matter is the ability to estimate how an application would behave in the face of physical changes or under, sometimes quite extreme, environmental conditions [3], [4], [5], [6]. In network systems for example, this would include the ability to anticipate possible bottlenecks, upgrades, infrastructure planning, resource provisioning, tolerance strategies, etc. These estimations can be obtained from in-operating systems, which can be stressed, therefore highlighting their limits.

In fact, stochastic models, such as *Petri Nets* extensions [7], [8], [9] provide a reasonably accurate alternative to estimate the performance of a system that is being deployed. This work aims to design a modeling structure capable of representing behavioral aspects of the real system in the future. Therefore, such model can be simulated and estimations can be statistically assessed. Based on this simulation model, estimations for multiples scenarios can be combined and obtained with fairly easily and quickly.

This paper extends our previous work [10], where we introduced a deployment approach to improve network performance in container-based clouds. We now exploit a novel *Stochastic Petri Net* model that can graphically and mathematically represent network infrastructures. Then, it can be systematically assessed without having to construct, modify, or test

978-1-5386-6950-1/18/$31.00 ©2018 IEEE

real configurations. Petri Nets has been used to analyze many different areas. In Computer Science, it has been successfully utilized to evaluate a range of communication infrastructures, including the approaches in [11], [12], [13], [14], [15], [16].

To the best of our knowledge, this is the first model based on Stochastic Petri Nets and Link Aggregation that has been proposed for effectively modeling, analyzing, and improving the performance of container-based cloud environments in emerging network-intensive distributed applications. In addition to providing a general model for deploying network infrastructures, we also show that this model can be simulated to estimate network performance in cloud-based systems. Also, it provides a suitable way to deploy network link aggregation.

Furthermore, we evaluate how network-intensive applications perform in a container-based cloud environment to demonstrate applicability, feasibility, efficiency, and accuracy. The combination of all these modeling and analysis features characterize the novel aspect of the paper. Because we exploit the design level of the network, implementation details are irrelevant and therefore analysis can be conducted without a complete version of the real system, which tends to be quite valuable when deploying prototypes.

The article is organized as follows: Section II introduces the background related to the literature in Section III; Section IV describes the proposed model, which is extended for network aggregations in Section V. Finally, Section VI discusses our conclusions and future work.

II. Background

A. Petri Nets

An option widely used for in-advance assessment of dynamic systems, including networked cloud-based systems [5], [14], [17], [18] is given by *Petri Nets* (PNs) [7], [8], [9]. PN structures a powerful formalism combining mathematical and intuitive modeling interfaces. It therefore provides mechanisms to model and assess systems characterized by concurrency, synchronization, resource sharing, faults, failures, bottlenecks, and others [12], [19]. These features appear quite often in advanced networked systems, which makes PNs a natural choice for modeling them.

Structurally, a PN is composed of *places* (modeling states), *transitions* (modeling state changes), and *directed arcs* (connecting places and transitions). To express the conditions that hold in a given state, places are marked with *tokens*.

Extensions of PNs have been developed to represent an important class of time-dependent processes, such as communication channels, code processing, hardware designs, and system workflows.

Generalized Stochastic Petri Nets (GSPNs) [8], for example, is an extension that combines timed and non-timed PNs. In GSPN, *time* is represented by random variable, exponentially distributed, which are associated to *timed transitions*. When the time is irrelevant for a given transition, one can simply use *non-timed* (or *immediate*) transitions.

Formally, a GSPN is a 7-tuple $GSPN = \langle P, \mathcal{T}, \Pi, I, O, M, W \rangle$, where:

- $P = \{p_1, p_2, \ldots, p_n\}$ is a finite set of places;
- $\mathcal{T} = \{t_1, t_2, \ldots, t_m\}$ is a finite set of transitions;
- $\Pi : \mathcal{T} \rightarrow \mathbb{N}$ is the priority function, where:

$$\Pi(t) = \begin{cases} \geq 1, & \text{if } t \in \mathcal{T} \text{ is immediate;} \\ 0, & \text{if } t \in \mathcal{T} \text{ is timed.} \end{cases}$$

- $I : (\mathcal{T} \times P) \rightarrow \mathbb{N}$ is the input function that defines the multiplicities of directed arcs from places to transitions;
- $O : (\mathcal{T} \times P) \rightarrow \mathbb{N}$ is the output function that defines the multiplicities of directed arcs from transitions to places;
- $M : P \rightarrow \mathbb{N}$ is the initial marking function. M indicates the number of tokens[1] in each place, i.e., it defines the state of a GSPN model;
- $W : \mathcal{T} \rightarrow \mathbb{R}^+$ is the weight function that represents either the immediate transitions weights (w_t) or the timed transitions rates (λ_t), where:

$$W(t) = \begin{cases} w_t \geq 0, & \text{if } t \in \mathcal{T} \text{ is immediate;} \\ \lambda_t > 0, & \text{if } t \in \mathcal{T} \text{ is timed.} \end{cases}$$

The relationship between places and transitions is established by the sets ${}^\bullet t$ and $t^\bullet$, defined as follows.

Definition 1: Given a transition $t \in \mathcal{T}$, define:

- ${}^\bullet t = \{p \in P \mid I(t,p) > 0\}$ as the *pre-conditions* of t;
- $t^\bullet = \{p \in P \mid O(t,p) > 0\}$ as the *post-conditions* of t.

A state of a GSPN changes when an enabled transition fires. Only enabled transitions can fire. *Immediate* transitions fire as soon as they are enabled. The *enabling* and *firing* rules for transitions are defined next.

Definition 2 (Enabling Rule): A transition $t \in \mathcal{T}$ is said to be enabled in a marking M if and only if:

- $\forall p \in {}^\bullet t, M(p) \geq I(t,p)$.

When an enabled transition fires, it removes tokens from input to output (its *pre* and *post* conditions).

Definition 3 (Firing Rule): The firing of transition $t \in \mathcal{T}$ enabled in the marking M leads to a new marking M' such that $\forall p \in ({}^\bullet t \cup t^\bullet)$, $M'(p) = M(p) - I(t,p) + O(t,p)$.

A GSPN is said to be *bounded* if there is a limit $k > 0$ on the number of tokens in each place. Therefore, it ensures that the state-space resulting from a bounded GSPN is finite.

When the number of tokens in each input place p of t is N times the minimum needed to enable t ($\forall p \in {}^\bullet t$, $M(p) \geq N \times I(t,p)$, where $N \in \mathbb{N}$ and $N > 1$), it enables the transition to fire more than once. In this situation, the transition t is said to be enabled with degree $N > 0$. Transition firing may use one of the following dynamic semantics:

- *single-server*: N sequential fires;
- *infinite-server*: N parallel fires;
- *k-server*: the transition is enabled up to k times in parallel; tokens that enable the transition to a degree higher than k are handled after the first k firings.

GSPNs can be isomorphic according to *Continuous-Time Markov Chains* (CTMC) [8]. However, they are more expressive, because they allow metrics to be computed by

[1]Black dots are usually used to graphically represent a token in a place.

both simulation and analysis of the state-space. In the last case, GSPN are indeed converted into CTMC for analysis. Furthermore, GSPNs allow to combine exponential arranges to model different time distributions [20].

B. Link Aggregation

Link aggregation, also referred to as trunking, is a technology defined in the IEEE 802.1AX-2014 Link Aggregation Standard (formerly IEEE 802.3ad) [21]. The standard is a layer 2 control protocol that provides a method to combine the capacity of multiple full-duplex Ethernet links into a single logical link. This link aggregation group is then treated as if it were in fact a single link. The main benefits of link aggregation are [21]:

- *Increased bandwidth:* multiple links combined into one logical link;
- *Automatic failover and failback:* traffic from a failed link is automatically switched over to other links in the aggregation;
- *Improved administration:* all interfaces are administered as a single unit.
- *Less drain on the network address pool:* all aggregation can be assigned one IP address.

In order to use the basic link aggregation, the respective network devices must first be configured so that the interfaces are aggregated as one group. There are two distinct methods to configure basic link aggregation: static or dynamic. In the static method, all configuration settings are setup directly on each participating group's network device. In the dynamic method, each function begins dynamically using the Link Aggregation Control Protocol (LACP).

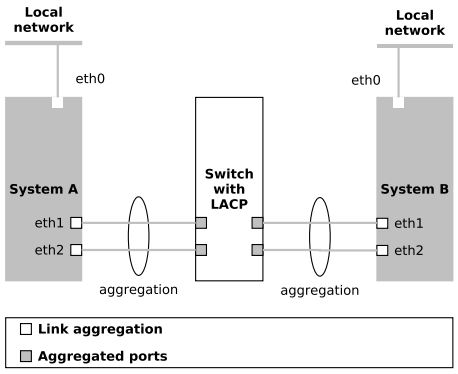

Fig. 1. Traditional link aggregation topology using a switch with LACP.

The dynamic method ensures that the protocol automatically detects interface failures and performs failover to standby interfaces. A failure is assumed and the relevant network device becomes unavailable. As a result, the sending or receiving of packets is only performed by the remaining interfaces. In the static method on the other hand, interface failures must be manually identified when they occur above the physical level or between peers that are not directly connected. The disadvantage here is that there is no method to detect any kind of cabling or configuration errors.

Figure 1 illustrates a local network with two systems. Each system has a link aggregation configured using the dynamic method. Note that the two systems are connected by a switch with LACP. System A has an aggregation that consists of two interfaces, eth1 and eth2. These interfaces are connected to the switch through aggregated ports. System B also has an aggregation of two interfaces, eth1 and eth2. These interfaces are also connected to aggregated ports on the switch. In this link aggregation topology, the switch must support the IEEE 802.1AX-2014 standard and the switch ports must be configured for aggregation.

C. Container-based Cloud

The container-based virtualization approach is also known as *operating system level* virtualization. Here the virtualization layer runs as an application within the operating system. The operating system kernel runs on the hardware host with several isolated VM guests called *containers*.

Technically, container-based virtualization is a lightweight alternative to *hypervisors* [22]. This approach splits the physical machine resources, creating multiple isolated user space instances on the same operating system. Users have the illusion that they are working on their own independent network, memory, and file system. Figure 2 presents a high-level configuration of the traditional architecture of Linux containers (LXC).

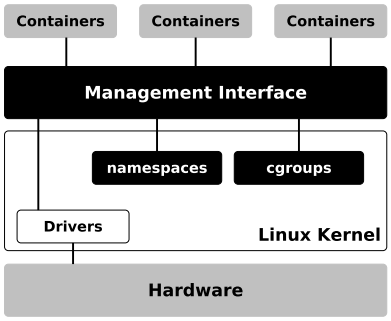

Fig. 2. This illustrates the architecture of LXC.

It must be emphasized that LXC basically uses two kernel features to define processes [23]: *namespaces* and *cgroups (control groups)*. Kernel namespaces provide process isolation by creating separate namespaces for containers and enable the creation of an abstraction of a particular global system resource and make it appear as a separated instance. As a result, multiple containers can use the same resource simultaneously without creating a conflict. The kernel also uses cgroups to group processes to manage system resources. Moreover, cgroups allows for the allocation of CPU time, system memory, and network bandwidth among user-defined groups of tasks. Finally, the management interface creates a higher layer that interacts with the kernel components and provides tools for constructing and managing containers.

In short, LXC runs at the operating system level, providing abstractions directly for guest applications. Thus, all containers share a single operating system kernel and should have weaker isolation than hypervisor-based systems. However, from the customer's point of view, each container executes exactly as if it were a stand-alone operating system.

LXC is currently an attractive lightweight virtualization in cloud environments. It runs a distinct image that provides all files necessary to support the processes, because it is included in the Linux kernel. Thus, it allows a set of processes to be isolated from the rest of the system, providing an image that contains all of an applications dependencies. Moreover, it is portable and consistent as it moves from development, to testing, and finally to production.

III. Related Work

This section presents an overview on the research related to the topics addressed in this paper, as shown in the Figure 3.

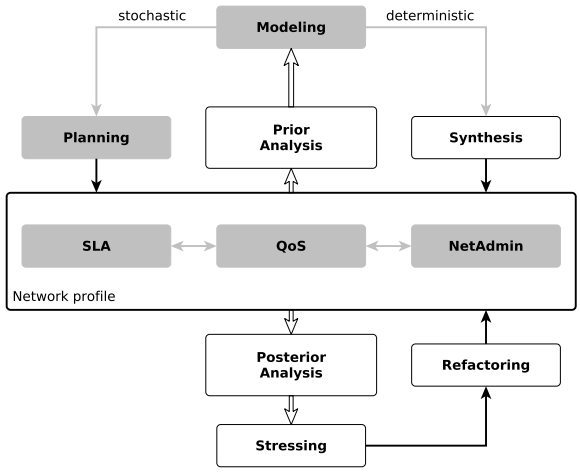

Fig. 3. Schematic literature overview.

A networked system includes at least three central interconnected components: SLA regulator, QoS monitor, and networking engine. Each of these components can be approached at many different abstraction levels, but in general two high-level strategies can be identified in the literature:

(a) *a posteriori* analysis, which is a *top-down* view that waits for the infrastructure to be developed, so it can be monitored, stressed, measured, and subsequently improved [13], [15], [17], [24];

(b) *a priori* analysis, which defines a *bottom-up* view that tries to understand predictably, how the system will behave under certain conditions, and then based on this it implements proactive functional corrections [11], [16], [25], [26].

Posterior analysis is focused more on monitoring and controlling the relationship of the network partners. It has a wide range of tools to support it and a massive range of verification methods and techniques. Hence, the infrastructure already exists, so possible improvements primarily depend on re-factoring.

On the other hand, prior analysis aims to anticipate how the network system would behave under particular variable conditions. This is a predictive approach that can be conducted anytime during the system's life-cycle. For example, if the infrastructure already exists, it can be useful for capacity planning, resource allocation, elasticity, etc.. Moreover, it can also be useful for planning the construction of the system itself. In any case, this approach requires a model that is able to reproduce the network behavior with a certain level of abstraction, in order to properly conduct experiments.

Models and prototypes have been extensively used in the literature to formalize and investigate many different aspects of network systems in cloud environments [10], [14], [16], [18]. One can specifically identify two classes of predictive analysis models: *Stochastic* and *deterministic* models [27].

Stochastic analysis is a wide term to indicate the combination and interaction between classical analysis and the theory of probability. Thus, *Petri Nets* allow engineers to create a formal model to estimate resource consumption for a range of workload profiles and network configuration options, within a short period of time [11], [28], [29]. This enables realistic SLA clauses with an agreement between customers and providers, in addition to helping to plan system upgrades, discovering or preventing bottlenecks, and anticipating potential SLA violations.

In this paper, we are more interested on the stochastic level of modeling, as highlighted in Figure3. Although the literature provides some options for predictive stochastic analysis of networks, we argue that our proposal is innovative because it takes parallel advantages from *Link Aggregation* (Section II-B) and *Container-based Cloud* (Section II-C). These are integrated in a cloud computing environment in order to provide the opportunity to manage the trade-offs between scale-up and scale-out. For instance, in the bare-metal scenario, the size of each server is defined by the available hardware. Consequently, the system administrator has to adjust the application to fit this size. On the other hand, when system administrators are in a cloud scenario, they can reconfigure the amount of resources based on the needs of the application. This feature allows the administrator to optimize resource usage while delivering better completion times.

In addition to the advantages of the cloud computing environment, we are also interested in reducing the performance degradation that is commonly caused by using virtualization technologies. Therefore, we use *Linux Containers* in our proposed model, because container-based instances provide almost the same performance as the native environment [30].

Finally, we also show how network bandwidth can be increased by using the Link Aggregation [21], which results higher availability and capacity, while the network infrastructure remains unchanged.

IV. Proposed model for regular clustering

Now we introduce the GSPN model proposed in this paper to estimate network bandwidth utilization under different system setups. Based on the model's estimations, we then show how the system can be modified to create performance improvements and realistic infrastructure planning. Our approach verifies throughput and latency in addition to comparing the overhead added by using container-based cloud instances. We used the NetPipe benchmark [31] in two bare-metal configurations (Regular TCP and IEEE 802.1AX-2014) and a cloud instance configuration with both LXC and IEEE 802.1AX-2014 deployed.

We begin by modeling the regular (non-aggregated) setup and then we extend it to consider link aggregation in the next section. Consider the GSPN model shown in Figure 4.

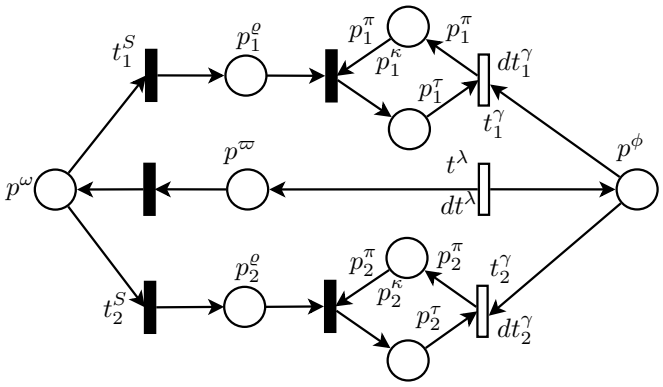

Fig. 4. Proposed model for the regular scenario.

The model starts when the timed transition t^λ *fires* tokens toward a *master* place named p^ϖ, representing network requests arriving for processing. Tokens are fired according to the delay dt^λ of the transition t^λ. Place p^ϖ is responsible for centralizing these requests until they can be distributed over the network by a switch device, modeled by the place p^ω. We consider the transition from p^ϖ to p^ω as immediate.

Requests in p^ω can be served (executed) either by the *slave* processor t_1^S or t_2^S, with the same probability. Once a slave service is chosen, the respective token (request) remains in a buffer named p_i^o $i = 1, 2$, while it is waiting to be served by the slave.

This buffering time is immediate, as long as the slave has enough resources (network bandwidth) to transmit the request. Otherwise it has to wait until this is the case. The calculation of the enabling rule is: p_i^o is served whenever the number of resources available (place p_i^κ) is equal to or greater than the the package size to be transmitted (arcs weight from/to p_i^κ), i.e., $p_i^\pi < \#p_i^\kappa$.

After being served (place p_i^τ) during a certain time (transmission time), which is defined by the delay of the timed transition t_i^γ (dt_i^γ), the token leaves the model, which means that the request answer has been replied by the opposite link channel, which we do not aim to model here. The delay dt_i^γ is derived from the latency (ping time) added by the available bandwidth from the message pack mean size, i.e.,

$$dt_i^\gamma = latency + \frac{bandwidth}{mean\ message\ size}.$$

A. Experiment Setup

A set of input parameters were required for the GSPN in Figure 4 to be simulated. We specifically provided the setup parameters in Table I.

TABLE I
Model setup parameters

dt^λ	$\#p_i^\kappa$	p_i^π	dt_i^γ
600	1048576	100 to 800	$lat + \frac{\#p_i^\kappa}{p_i^\pi}$

The delay dt^λ of the transition t^λ defines the desired request arrival rate. In general, it can be switched so that the different workloads are simulated. In our experiment, we initially kept dt^λ fixed, because our main objective is to test network traffic (messages) load. In Section V we further variate both work and traffic loads.

We start by assigning a delay of 600 (ms) to dt^λ, which leads to the arrival rate of 0.6, i.e., close to 2 requests per second (req/s). Because our GSPN model has two slave modules to process master requests, the arrival rate for each slave module is close to $1/s$ which coincides to the arrival rate of the benchmark that is used.

Additionally, we set up the parameter $\#p_i^\kappa$ (second column) to 1048576, which corresponds to the network bandwidth (1 gigabits) in kB. Parameter p_i^π models message size and it is associated to transition weights that manipulate resources from/to $\#p_i^\kappa$. They are defined as a range, because they correspond exactly to the parameters we aim to variate. In this paper, we use a message size variation from 100 to 800 kB, which keeps us in accordance to the benchmark-guided experiments. This additionally leads us to assign the delay dt_i^γ, modeling the native network latency to transfer a message pack from one network point to another.

B. Statistical Results

If one has to estimate the network bandwidth utilization for a range of message sizes, let us assume that messages variate from 100 to 800 kB, as the parameters were established in Table I. A direct way to plot network saturation points for this range is by submitting the system to loading stress tests. In fact, we can use the Netpipe benchmark [31] to describe this online test, and the result is shown in Table II, row *Measured*.

Netpipe [31] is a protocol independent performance benchmark that visually represents the network performance under a variety of conditions. It uses a simple series of ping-pong tests over a range of message sizes to provide a complete measurement of network performance. Message sizes are chosen at regular intervals with slight differences to provide a complete evaluation of the communication system. Each data point includes many ping-pong tests to provide accurate

timing. Latency is calculated by dividing the round trip time in half for small messages (less than 64 Bytes).

On the other hand, those saturation points could be anticipated by simulating the proposed GSPN model in Figure 4. The advantage would be that one could simulate, within a reasonable accuracy, a much larger range of points, in a much shorter period of time, with no implementation costs. In addition, it enables multiple analyses to be made, such as local saturation points, bottlenecks, etc. By simulating our model to the range of message sizes, we obtained the results shown in Table II, row *Estimated*.

TABLE II
SYSTEM UTILIZATION (REGULAR): MEASURED AND ESTIMATED COMPARISON

	System Utilization								
p_i^{π}	100	200	300	400	500	600	700	800	Avg
Measured	88%	89%	90%	90%	91%	91%	91%	92%	90,1%
Estimated	90%	93%	94%	94%	96%	98%	99%	99%	95,4%

It is important to note that when the transition t^{λ} first fires, in addition to inserting a token in p^{ϖ}, it also creates a copy of this token in p^{ϕ}. Therefore, by collecting statistics in p^{ϕ}, we are able to describe the entire system's expected behavior. Because we are assuming the same probability of the slaves to serve and also the same standard for message sizes, we only focus on monolithic analysis. To estimate system utilization in Table II, we implemented the formula $P\{\#p^{\phi} > 0\}$ using the *TineNET* tool [32] which computes the probability for tokens in p^{ϕ}. In general, our estimations show an accuracy level of 95.4%, which we argue is reasonable from a stochastic point of view.

V. EXTENDED GSPN MODEL FOR AGGREGATED CLUSTERING

Now, we use the idea of *link aggregation*, presented in Section II-B, in order to extend our model to estimate how much this strategy improves the standard approach. Then we compare our estimations to the measured improvements again to check accuracy. Consider the GSPN model in Figure 5.

This model corresponds to a new version of the GSPN shown in Figure 4, extended to an additional structure that duplicates the processing power of each slave, as a result of the duplication of the real network link. Actually, the model can be generically replicated to any number of network interfaces.

TABLE III
SYSTEM UTILIZATION (DUAL AGGREGATION): MEASURED AND ESTIMATED COMPARISON

p_i^{π}	100	200	300	400	500	600	700	800
Measured (Aggr.)	68%	75%	79%	82%	88%	88%	90%	91%
Estimated (Aggr.)	70%	78%	84%	86%	90%	89%	92%	94%
Estimated (Reg.)	90%	93%	94%	94%	96%	98%	99%	99%
Improvement	10%							

The impact of this strategy in real network systems remains to be seen. To do this estimation we fed our new model the same parameters as in Table I and then we did a simulation using the same setup for the range of network messages. As a result, we collected the estimations about system utilization for this new scenario. Table III shows the results, compares them to the system utilization estimated from the *regular* scenario, and to the system utilization measured from the real system running under the link aggregation setup.

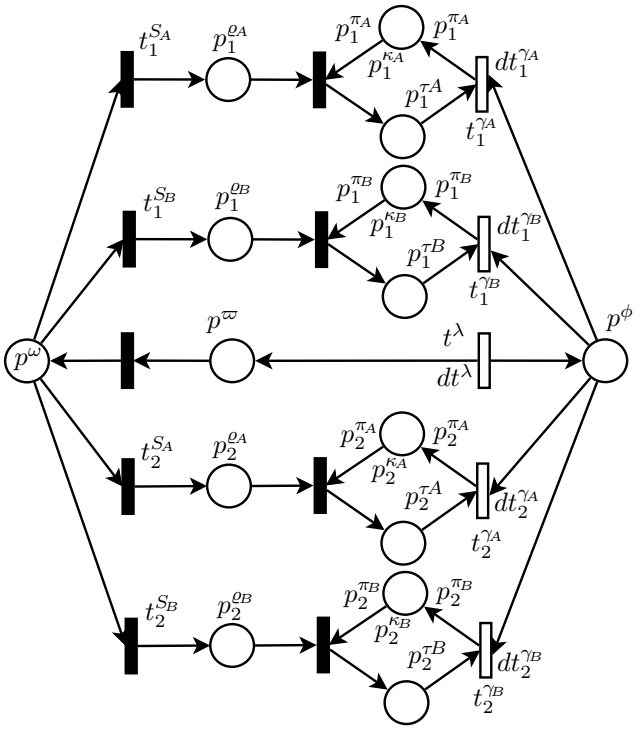

Fig. 5. Extended model including link aggregation.

There is an improvement around 10% from link aggregation to the regular setup. Although this may seem to be a minor improvement, it has to be highlighted that the first message size (100kB) was 20% less saturated. For the heaviest experiment (800kB), the difference was smaller. Even though link aggregation is not at the saturation level (94%), the regular setup is already far saturated. Actually it saturates much earlier, with traffic of 600kB messages.

Thus, from the presented experiment, we could say that we did not find the saturation point for (dual) link aggregation.

VI. CONCLUSION

In this paper, we proposed a model by using *Stochastic Petri Net* approach to evaluate, estimate, and improve network bandwidth utilization under different system setups. Our model anticipates the overhead imposed by adding container-based cloud instances, in the context of LXC and IEEE 802.1AX-2014 link aggregation solutions. The main goal was to increase the network bandwidth and, consequently, achieve better throughput and latency.

In addition to the proposed modeling structure, we also conducted a set of experiments in order to asses a number of different system setups. We subsequently constructed a real network infrastructure based on the model's suggestions,

attempting to show how estimations would impact system performance in practice. We used the standard industry benchmark *NetPipe* in two bare-metal configurations (Regular TCP and IEEE 802.1AX-2014) and a cloud instance configuration with LXC and IEEE 802.1AX-2014 deployed.

Results revealed that the model can anticipate, with reasonable accuracy, the structural and behavioral characteristics of cloud computing. Hence, network efficiency can be improved systematically without the need for complete real infrastructure modifications. It is also important to highlight that other network-intensive applications could take advantage of our model, such as big-data applications that require high bandwidth and throughput as well as low latency [33], [34].

In the future, we hope to provide a framework that can be easily integrated with cloud platforms (*e.g.*, OpenStack and OpenNebula) to evaluate, estimate, and enhance network bandwidth in an elastic way, without the need for real infrastructures.

ACKNOWLEDGMENT

The authors would like to thank CAPES, FAPERGS, PUCRS and SETREM for their partial financial support.

REFERENCES

[1] P. M. Mell and T. Grance, "Sp 800-145. the nist definition of cloud computing," Gaithersburg, MD, United States, Tech. Rep., 2011.

[2] G. Wang and T. S. E. Ng, "The impact of virtualization on network performance of amazon ec2 data center," in *2010 Proceedings IEEE INFOCOM*, March 2010, pp. 1–9.

[3] G. Kousiouris, A. Menychtas, D. Kyriazis, S. Gogouvitis, and T. Varvarigou, "Dynamic, behavioral-based estimation of resource provisioning based on high-level application terms in cloud platforms," *Future Generation Computer Systems*, vol. 32, pp. 27 – 40, 2014.

[4] S. Wandelt, X. Sun, M. Zanin, and S. Havlin, "Qre: Quick robustness estimation for large complex networks," *Future Generation Computer Systems*, pp. –, 2017.

[5] J. P. Macker and I. Taylor, "Orchestration and analysis of decentralized workflows within heterogeneous networking infrastructures," *Future Generation Computer Systems*, vol. 75, pp. 388 – 401, 2017.

[6] A. M. Chirkin, A. S. Belloum, S. V. Kovalchuk, M. X. Makkes, M. A. Melnik, A. A. Visheratin, and D. A. Nasonov, "Execution time estimation for workflow scheduling," *Future Generation Computer Systems*, vol. 75, pp. 376 – 387, 2017.

[7] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE, v.77*, pp. 541–580, 1989.

[8] D. Kartson, G. Balbo, S. Donatelli, G. Franceschinis, and G. Conte, *Modelling with Generalized Stochastic Petri Nets*, 1st ed. John Wiley & Sons, Inc., 1995.

[9] M. A. Marsan, G. Balbo, and G. Conte, "A class of generalized stochastic Petri nets for the performance analysis of multiprocessor systems," in *ACM Transactions on Computer Systems*, vol. 2, 1984, pp. 1–11.

[10] C. Rista, D. Griebler, C. A. F. Maron, and L. G. Fernandes, "Improving the network performance of a container-based cloud environment for hadoop systems," in *2017 International Conference on High Performance Computing Simulation (HPCS)*, July 2017.

[11] M. Teixeira, R. Ribeiro, C. Oliveira, and R. Massa, "A quality-driven approach for resources planning in service-oriented architectures," *Expert Systems with Applications*, vol. 42, no. 12, pp. 5366 – 5379, 2015.

[12] R. German, *Performance Analysis of Communication Systems : Modeling with Non-Markovian Stochastic Petri Nets*, 1st ed. United Kingdom: John Wiley and Sons, 2000.

[13] O. J. Haggarty, W. J. Knottenbelt, and J. T. Bradley, "Distributed response time analysis of gspn models with mapreduce," in *2008 International Symposium on Performance Evaluation of Computer and Telecommunication Systems*, June 2008, pp. 82–90.

[14] H. He, S. Pang, and Z. Zhao, "Dynamic scalable stochastic petri net: A novel model for designing and analysis of resource scheduling in cloud computing," *Scientific Programming*, vol. 2016, 2016.

[15] Y. Yagawa, A. Sutoh, E. Malamura, and T. Murata, "Modeling and performance evaluation of cloud on-ramp by utilizing a stochastic petri-net," in *IIAI International Congress on Advanced Applied Informatics*, July 2016, pp. 995–1000.

[16] H. Chen, C. Zhou, Y. Qin, A. Vandenberg, A. V. Vasilakos, and N. Xiong, "Petri net modeling of the reconfigurable protocol stack for cloud computing control systems," in *IEEE International Conference on Cloud Computing Technology and Science*, Nov 2010, pp. 393–400.

[17] G. Fan, H. Yu, and L. Chen, "A formal aspect-oriented method for modeling and analyzing adaptive resource scheduling in cloud computing," *IEEE Transactions on Network and Service Management*, vol. 13, no. 2, pp. 281–294, June 2016.

[18] Y. Cao, H. Lu, X. Shi, and P. Duan, *Evaluation Model of the Cloud Systems Based on Queuing Petri Net*. Cham: Springer International Publishing, 2015, pp. 413–423.

[19] M. Teixeira, R. Ribeiro, C. Oliveira, and R. Massa, "A quality-driven approach for resources planning in service-oriented architectures," *Expert Systems with Applications*, vol. 42, no. 12, pp. 5366 – 5379, 2015.

[20] A. A. Desrochers, *Applications of Petri Nets in Manufacturing Systems: Modeling, Control and Performance Analysis*. IEEE Press, 1994.

[21] "IEEE standard for local and metropolitan area networks – link aggregation," *IEEE Std 802.1AX-2014 (Revision of IEEE Std 802.1AX-2008)*, pp. 1–344, Dec 2014.

[22] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. F. D. Rose, "Performance evaluation of container-based virtualization for high performance computing environments," in *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, Feb 2013, pp. 233–240.

[23] A. Vogel, D. Griebler, C. Schepke, and L. G. Fernandes, "An intra-cloud networking performance evaluation on cloudstack environment," in *25th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. St. Petersburg, Russia: IEEE, 2017, p. 5.

[24] A. G. Kumbhare, Y. Simmhan, M. Frincu, and V. K. Prasanna, "Reactive resource provisioning heuristics for dynamic dataflows on cloud infrastructure," *IEEE Transactions on Cloud Computing*, vol. 3, no. 2, pp. 105–118, April 2015.

[25] G. Mencagli, M. Vanneschi, and E. Vespa, "A cooperative predictive control approach to improve the reconfiguration stability of adaptive distributed parallel applications," *ACM Transactions Autonomic Adaptive Systems*, vol. 9, no. 1, pp. 2:1–2:27, Mar. 2014.

[26] A. Castiglione, M. Gribaudo, M. Iacono, and F. Palmieri, "Exploiting mean field analysis to model performances of big data architectures," *Future Generation Computer Systems*, vol. 37, pp. 203 – 211, 2014.

[27] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. New York: Springer Science, 2008.

[28] W. Tan, Y. Fan, and M. Zhou, "A petri net-based method for compatibility analysis and composition of web services in business process execution language." *IEEE Trans. on Automation Science and Engineering*, vol. 6, no. 1, pp. 94–106, 2009.

[29] W. M. P. Van der Aalst, "The application of petri nets to workflow management," *Journal of Circuits, Systems and Computers*, vol. 08, no. 01, pp. 21–66, 1998.

[30] R. Rizki, A. Rakhmatsyah, and M. A. Nugroho, "Performance Analysis of Container-Based Hadoop Cluster: OpenVZ and LXC," in *4th International Conference on Information and Communication Technology (ICoICT)*, May 2016, pp. 1–4.

[31] NetPipe, *A Network Protocol Independent Performance Evaluator (NetPipe)*, 2017. [Online]. Available: http://bitspjoule.org/netpipe

[32] A. Zimmermann, *TimeNET 4.1*, 2017. [Online]. Available: http://www.tu-ilmenau.de/TimeNET

[33] M. V. Neves, C. A. F. D. Rose, K. Katrinis, and H. Franke, "Pythia: Faster Big Data in Motion Through Predictive Software-Defined Network Optimization at Runtime," in *28th International Parallel and Distributed Processing Symposium (IPDPS)*, ser. IPDPS '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 82–90.

[34] R. F. E. Silva and P. M. Carpenter, "Controlling Network Latency in Mixed Hadoop Clusters: Do We Need Active Queue Management?" in *41st Conf. on Local Computer Networks*, Nov 2016, pp. 415–423.