Design and Evaluation of a Virtual Experimental Environment for Distributed Systems

Luc Sarzyniec, Tomasz Buchert, Emmanuel Jeanvoine, Lucas Nussbaum

Design and Evaluation of a Virtual Experimental Environment for Distributed Systems

Luc Sarzyniec, Tomasz Buchert, Emmanuel Jeanvoine, Lucas Nussbaum
Inria, Villers-lès-Nancy, F-54600, France
Université de Lorraine, LORIA, F-54500, France
CNRS, LORIA - UMR 7503, F-54500, France

***Abstract*—Between simulation and experiments on real-scale testbeds, the combined use of emulation and virtualization provide a useful alternative for performing experiments on distributed systems such as clusters, grids, cloud computing or P2P systems. In this paper, we present Distem, a software tool to build distributed virtual experimental environments. Using an homogenenous set of nodes, Distem emulates a platform composed of heterogeneous nodes (in terms of number and performance of CPU cores), connected to a virtual network described using a realistic topology model. Distem relies on LXC (Linux Containers), a low-overhead container-based virtualization solution, to achieve scalability and enable experiments with thousands of virtual nodes. Distem provides a set of user interfaces to accomodate different needs (command-line for interactive use, Ruby and REST APIs), is freely available and well documented. After a detailed description of Distem, we perform an experimental evaluation of several of its features.**

***Keywords*-experimentation; large-scale; emulation; virtualization; experimental validation**

I. Introduction

Distributed systems such as grids, clusters, peer-to-peer systems, high-performance supercomputers, cloud computing infrastructures or desktop computing environments, benefit of an ever increasing popularity nowadays. Computer scientists traditionally study their systems *a priori* by reasoning theoretically on the constituents and their interactions. But the complexity of these distributed systems and the applications that are executed on them make this methodology near to impossible, explaining that most of the studies are done *a posteriori* through experiments.

Three main methodologies exist to experiment with computer systems [1]: real-scale experiments, simulation and emulation. *Real-scale experiments* (or *in situ*) consists in executing the real application under study on an real-scale experimental platform. On the opposite, with *simulation*, both the application and the environments are replaced by models, and the interactions between both models are computed by a simulator. Both approaches have advantages and disadvantages. Simulation enables its users to perform experiments at very large scale (millions of nodes) with a very low cost, but does not allow the experimenter to execute a real application – a model needs to be used instead, which can be perceived as providing lower realism. Experimenting using a real testbed is usually considered as providing higher realism since real hardware and real applications are used, but does not allow the experimenter to change the experimental environment: experiments are limited to what the testbed provides.

The combined use of emulation and virtualization provides a very interesting alternative. Using an homogeneous set of nodes from a real testbed, one can use emulation to alter the characteristics and performance of nodes and network, similarly to what would be possible using simulation. Additionally, using virtualization to create virtual nodes enables experiments at a much larger scale.

This paper presents the design and validation of Distem, a software tool combining emulation and virtualization to build large-scale, distributed, virtual experimental environments.

The rest of the paper is structured as follows. Section II presents the main motivations and features of Distem, and describes its design choices and implementation. Then, Section III describes a set of experiments aiming at validating most aspects of Distem. In Section IV, Distem is compared to previous work related to virtualized or emulated experimental environments. Finally, in Section V, we conclude our findings and describe our plans for future work.

II. Design of Distem

The architecture of Distem is simple and leverages various freely available tools and technologies that proved successful on their own. This section serves as a detailed explanation of how they work together to achieve the goals of Distem.

Distem is open source software under the GPLv3 license and is available from http://distem.gforge.inria.fr.

A. Goals and features

Distem was conceived to ease the design of the experiment, offloading the experimenter from tedious and repeatable tasks, to give additional control over the configuration of the experimental infrastructure and, finally, to improve the quality of scientific results. To achieve these goals, various features are present in Distem.

Heterogeneity emulation: The most important feature of Distem is its ability to shape the environment to the requirements of the experiment. For example, the user can introduce heterogeneity of resources (network links, performance and number of CPUs) to otherwise homogeneous cluster. Although probably less useful in practice, the opposite should also possible, i.e., building homogeneous infrastructure out of heteregeneous components.

Many different emulation scenarios are possible: long distance networks (high latency), clusters (high CPU power, very low latency), grids (high CPU power, varying latency), peer-to-peer topologies or Internet (high heterogeneity), cloud computing (no strict guaranties on the resources, variable congestion of the network). All these situations can be readily represented in Distem. The parameters of the network and CPU speeds can be changed on-the-fly, allowing for modelling situations whose environment varies with time (e.g., aforementioned cloud computing).

A unique feature of Distem that is not present in other solutions, is integrated, fine-grained emulation of CPU resources. The physical cores on machines can be exclusively assigned to specific virtual node and can be tuned to run slower than their nominal speed. This may be crucial in some applications, especially ones that involve a large amount of computation. One has also to remember that if CPU emulation is in place, then the number of virtual nodes on a physical node is limited to the number of CPU cores it has.

Virtualization of the environment: Distem offers an easy way to start multiple, virtualized nodes on each physical node. First, it abstracts away from the location where the virtual nodes are actually deployed. Second, it automatically transfers images of the virtual nodes, taking care of caching and low-level details. Still, if there is a good reason to do so, one can manually specify where the virtual node will be deployed. This feature is very important, because configuring such a system on one's own is tedious and error-prone task. There are many details that must be taken care of and, if they are overlooked, wrong conclusions may be drawn from the obtained results.

Description of experiments: Distem paves a way to reproducible experiments, by encouraging researchers to clearly define their experimentation environment. Moreover, the programmatic interface of Distem makes it possible to write the experiments as Ruby programs, instead of the common approach of writing less maintainable shell scripts. Such programs may perform automatic analysis of the data obtained or change parameters of the environment, dynamically reacting to the events.

User friendliness: Distem also strives to be user-friendly. First, it offers a helping hand in defining the topology of the network, by automatically configuring routing tables in the nodes to route packets properly. Doing this manually would be tedious and complicated. Second, Distem offers three different user interfaces with increasing level of complexity and number of features. Depending on the situation and the experience of the user, one interface may be more appropriate than others. This also provides interoperability, since the low-level interface is programming language agnostic.

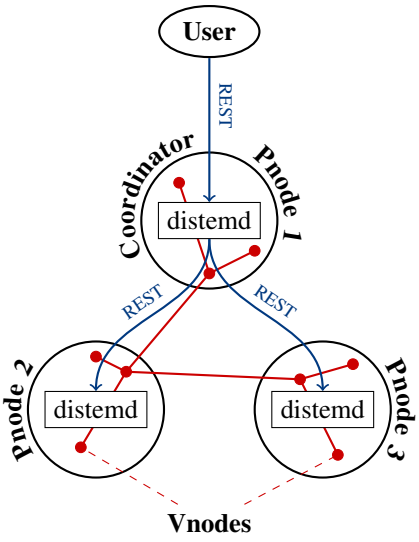

Figure 1. The communication architecture of Distem. The user created 3 physical nodes (*Pnodes*), each containing 3 virtual nodes (*Vnodes*). Every command the experimenter runs is first interpreted by the coordinator's instance of Distem (*distemd*) and then is routed to affected physical nodes in form of HTTP REST requests.

B. High-level architecture

Distem uses non-virtualized, physical nodes (*Pnodes*) as a base for its infrastructure. Every *Pnode* may contain multiple virtualized nodes (*Vnodes*). They are transparently separated and are not aware of each other's presence. All *Vnodes* may be connected by a virtualized Ethernet network to satisfy the requirements of the experiment to run.

The graphical representation of the architecture is presented in Figure 1. Every physical node hosts its own instance of Distem daemon (distemd) that controls virtual nodes hosted inside it, configuration of virtual network links and other resources assigned to the nodes. One of the nodes, called *coordinator*, is special, as it is responsible for controlling the whole experimental infrastructure, by communicating with the remaining nodes.

The user has to provide an image of the system that the virtual nodes will use. It is also possible to ask Distem to share one system image between the virtual nodes inside each physical node. That way one avoids a large part of the communication needed to distribute images and, additionally, can run more virtual nodes inside one physical machine.

One of the first decisions made during development of Distem was to use REST as a communication paradigm [2] with JSON as an intermediary representation of data. There are multiple reasons supporting this idea:

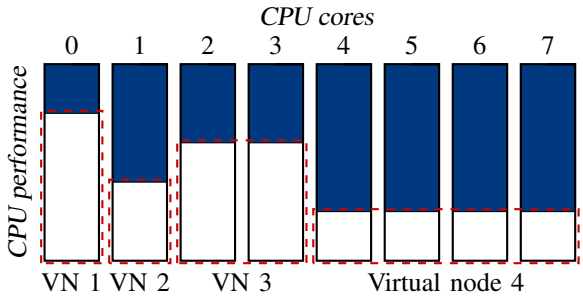

Figure 2. The emulation of CPU performance inside a single physical node. Each virtual node is given a different effective CPU speed, but the speed of each core inside the virtual nodes is the same.

1) It provides a clean and well-defined interface.
2) REST proved to be successful in modelling hierarchical structures like the architecture of Distem.
3) REST and JSON are programming language agnostic, but, on the other hand, are commonly supported by high-level languages.
4) REST and JSON are becoming a *de facto* standard of, respectively, architecture and serialization on the Internet.

That decision enabled us to create a well defined stack of interfaces to Distem, each built on top of the previous one:

1) REST interface – a well defined and structured schema to control the resources inside Distem.
2) Ruby interface – a programmatic way to work with Distem, by directly accessing the REST interface.
3) Command line interface – it leverages the Ruby library above to access Distem from the command line.

C. Node virtualization

The *Vnodes* inside Distem are Linux Containers (LXC). This is a very lightweight approach to virtualization, where the containers are given separate namespaces for system resources like tasks, network interfaces, memory, hard disks, etc. This feature was introduced in Linux kernel 2.6.27, released in October 2008.

However, by default the containers are not well separated – they share the same kernel version and, more importantly, the processor. To cope with this problem, we use Linux Control Groups, another prominent feature of Linux kernel. Using them, processor cores can be assigned to subset of processes in the system or, in our particular case, to the container.

D. CPU emulation

Linux Control Groups make it easy to assign computing cores to the containers, but in general this may not be enough. It is useful to control also the speed of the assigned cores. Whereas contemporary processors have some means to control their execution speed, they are insufficient, as they cannot be set to an arbitrary value. To overcome this problem, Distem uses one of the two algorithms to limit the speed of CPU cores: CPU-Hogs or CPU-Gov.

The first methods consists in scheduling real-time processes on the cores assigned to the node, and letting them consume configured amount of CPU time. The second one periodically changes the frequency of the processor cores so that in the long run the CPU computational power is as desired. As the second method depends highly on the hardware used in the experiment, the first method is prefered in Distem.

From the researcher's point of view, however, the effect is the same. The virtual nodes will run accordingly slower than the physical node, as has been presented in Figure 2.

More details and the evaluation of both methods were presented in [3].

E. Network emulation

Distem is able to precisely control the parameters of virtual network interfaces (addresses, network topology, latency and bandwidth). To achieve this, various features of Linux kernel are used together.

First, every virtual node can be equipped with multiple virtual network interfaces. They are implemented with *veth* interfaces (Virtual Ethernet device). Such a device transmits every packet from the physical node to the virtual node and vice-versa. The virtual devices of all virtual nodes are bridged together inside their common physical node. Finally, the bridge also contains the real physical link and so all the virtual nodes can access the physical network.

It must be noted that all IP networks share the same Ethernet network. This means, for example, that Ethernet broadcast frames would reach every single virtual node. This was not a problem in our experiments nor in our previous work with Distem, but nevertheless must be taken into a consideration. One of the ways to separate the IP networks in a more realistic way is to use virtual switch, e.g., VDE (Virtual Distributed Ethernet)[1] or Open vSwitch[2].

One of the problems that commonly arise when large, unsegmented networks are created (e.g., in cloud computing) is overflow of ARP caches in the operating systems. Linux, for example, by default will store up to 1024 entries in the cache. In fact this is a hard limit that can be reached only for a certain amount of time, before the tables will be flushed. To make it possible to deploy thousands of nodes, Distem raises the kernel limits automatically on every physical node.

To implement limitation of link latency and bandwidth, pluggable queueing disciplines in Linux kernel are used. More precisely, to implement emulation of the latency, *netem* (Network Emulation) queueing discipline is used. On the other hand, to limit the bandwidth of the link, TBF (Token

[1] http://vde.sourceforge.net/
[2] http://openvswitch.org/

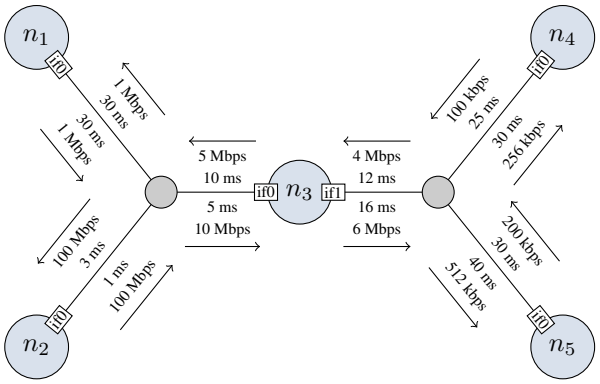

Figure 3. An example of a network topology that can be modeled in Distem. Five different virtual nodes are split into 2 IP networks and the third node serves as a gateway between them. Each link is configured with different latency and bandwidth. The *virtual switches* (smaller, gray nodes) are not actually present in the topology - they are merely a result of emulated links.

Bucket Filter) algorithm is used. In order to work around shortcomings of the Linux `tc` subsystem, we introduce IFB (Intermediate Functional Block) device to be able to apply limitations to incoming traffic as well.

When both latency and bandwidth limitation are in place, a special care must be taken to properly adjust the parameters of both or otherwise one can affect the other. For example, if latency is introduced and the associated buffer for the delayed packets is not large enough, they will be dropped by the kernel forcing retransmissions of packets in the higher protocols and consequently the bandwidth will become limited as well. Fortunately, Distem handles this case internally and the user does not have to do it manually.

In Figure 3, a typical example of topology emulated by Distem is presented.

III. Experimental validation

There are 2 main features of Distem that require testing: network emulation and CPU emulation. Moreover, since Distem was conceived as a tool to run large-scale experiments on commodity hardware and with limited number of physical machines, one has to evaluate also scalability of Distem deployment procedure.

All experiments were conducted using the Grid'5000 platform.

A. Network emulation

Distem is able to configure two different parameters of network links: link latency and link bandwidth. Moreover, the limitation can be applied to both outgoing and incoming packets independently. The details how the emulation is implemented vary slightly in the two cases as has been described in Section II-E, and as a result some experiment in this section has been carried out in both situations.

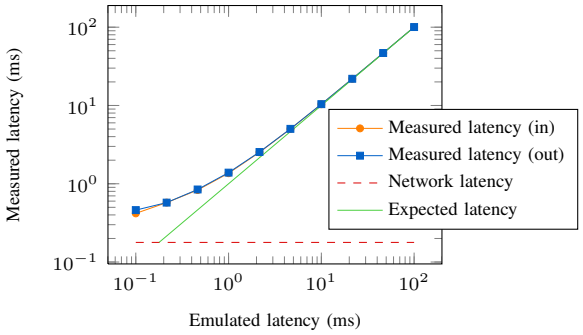

Figure 4. Emulation of network latency (logarithmic plot). The measured latency in Distem closely approximates the expected outcome.

For the results below, if applicable, 95% confidence intervals (using t-Student distribution) were computed and are presented together with the results.

1) Latency emulation: In the first experiment we measure the latency of the emulated link, to see if it corresponds to the emulated value. To this end, we use Distem to configure 2 virtual nodes: the first one sends UDP packets and the second sends it back upon reception. The time required for a packet to return (round trip time) is a single measurement. This value also includes the physical latency of the network and the small amount of CPU time required to process the packets on both ends. This can be neglected as we can measure this added value beforehand, without any emulation.

The results of this experiment are shown in Figure 4. Every measurement was repeated 30 times and then averaged. The measured latency of the physical link was 0.18 ms. The emulation gives precise results for values slightly larger than this value, i.e., values above 1 ms. In this experiment, latency has been emulated with both ingoing and outgoing limitations. As one can see in the plot, differences were negligible.

2) Latency emulation over time: The second experiment in this subsection concerns the quality of network emulation as offered by Linux operating system. In [4] it has been shown that the measured latency of the emulated link in older versions of Linux may depend on the clock interrupt frequency. This has been improved in more recent version of Linux kernel with the introduction of high frequency timers.

To reproduce the previous result, two virtual nodes are created in Distem. One of them starts a customized version of the `ping` program. It sends ICMP Echo packets in very regular intervals and measures the response time. If the emulated latency in Linux depends on the clock frequency, one should observe some inconsistent measurements during the time between interrupts, i.e., 4 miliseconds (default value on the x86 architecture).

In Figure 5, the measurement of latency during a period of 4 miliseconds is presented. As can be seen, the latency

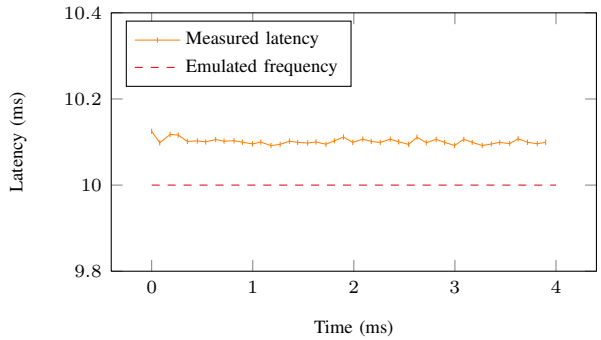

Figure 5. The latency of network link is constant throughout the measurement.

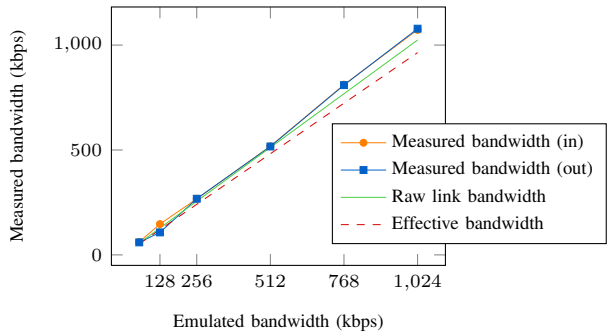

Figure 6. Emulation of low-bandwidth network link.

remains constant during this period, just as expected. The small gap (around 1.0% of the emulated latency) between measurement and the emulated value is caused by the physical latency of the link and processing time.

3) Bandwidth emulation: The next experiment evaluates the quality of network bandwidth emulation. Again, two virtual nodes are created with Distem and one of them has its bandwidth limited. Both ingoing and outgoing limitations have been tested. The bandwidth between these nodes is measured using the *iperf* benchmark (30 times for every bandwidth value) and compared with the intended value. The measured bandwidth is actually a TCP stream, therefore we also plot the maximum effective bandwidth, which equals roughly 94.15 % of the raw bandwidth value (TCP protocol with timestamps over IPv4). The measured, emulated value is expected to follow this trend.

The results are presented in Figure 6 (low bandwidth emulation) and Figure 7 (high bandwidth emulation). The range of values was chosen to represent real life computer links, like analog modems, ADSL connections and variations of Ethernet standard.

Clearly, the emulation of link bandwidth is correct. The results are stable, especially for large values of emulated bandwidth.

4) Simple topology: In this experiment we implement in Distem the topology presented in Figure 3. Then we measure

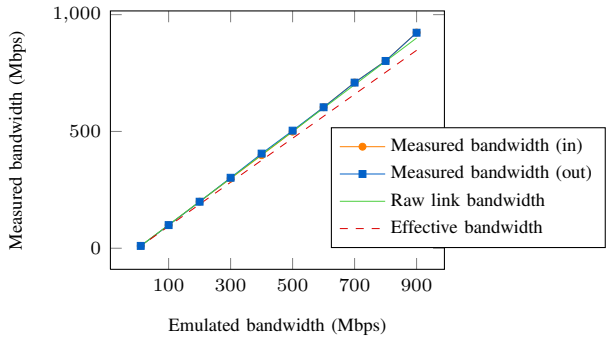

Figure 7. Emulation of high-bandwidth network link.

bandwidth and round-trip time between each pair of nodes to see whether the emulated topology behaves accordingly. Moreover, to ease the configuration phase, Distem was used to automatically choose gateway nodes and populate routing tables of nodes. Table I contains the summary of the results.

As can be easily seen, the results agree with the defined topology. In particular, the round-trip time of packets is a sum of latency emulated on every link the packet travels through, and the maximum bandwidth between two nodes is limited by the bottleneck link, i.e., the link with the minimum emulated bandwidth. This also explains why the round-trip time is symmetrical between any two nodes, but the bandwidth is not.

5) Analysis of Scp and Rsync: The next experiment was designed to test both emulation of latency and bandwidth, at the same time. Ideally, these parameters of the network link are independent, but, as was discussed above in Section II-E - they are fundamentally related. The experiment evaluates performance of two tools commonly used to copy files over network: `scp` and `rsync`. To see how network properties affect their performance, they are run under various pairs of latency and bandwidth values. Each run consists in copying the Distem 0.7 sources (115 files, about 700 KB) over the network.

The results are shown as 3D graph in Figure 8. A few interesting observations can be made. First, it can be seen that `rsync` outperforms `scp`, almost in all cases. The only case where the performance of `scp` reaches the speed of `rsync`, is when the latency of network link is close to zero. Moreover, we see that for both tools the time required to copy files is roughly inversely proportional to the bandwidth, which is expected. Finally, both programs need linearly more time to complete with the increasing latency value. This can be seen in Figure 10, which again reuses data from Figure 8. Both observed phenomena persist for any constant value of latency or bandwidth.

B. CPU emulation

In this experiment, we evaluate the CPU emulation features implemented in Distem. To this end, we executed High-

From \ To	n1	n2	n3	n4	n5
n1	-	0.06 s / 1.07 Mbps	0.08 s / 1.07 Mbps	0.16 s / 0.29 Mbps	0.17 s / 0.55 Mbps
n2	0.06 s / 1.04 Mbps	-	0.02 s / 9.57 Mbps	0.10 s / 0.29 Mbps	0.12 s / 0.55 Mbps
n3	0.08 s / 1.05 Mbps	0.02 s / 4.92 Mbps	-	0.08 s / 0.30 Mbps	0.10 s / 0.57 Mbps
n4	0.16 s / 0.17 Mbps	0.10 s / 0.16 Mbps	0.08 s / 0.15 Mbps	-	0.13 s / 0.15 Mbps
n5	0.17 s / 0.26 Mbps	0.12 s / 0.27 Mbps	0.10 s / 0.27 Mbps	0.13 s / 0.26 Mbps	-

Table I
ROUND-TRIP TIME AND BANDWIDTH IN THE TOPOLOGY PRESENTED IN FIGURE 3. THE TABLE CONTAINS RESULTS FOR EVERY PAIR OF NODES. FOR EXAMPLE THE ROUND TRIP TIME AND BANDWIDTH BETWEEN *n2* AND *n3* EQUALS 0.02 S AND 9.57 MBPS, RESPECTIVELY.

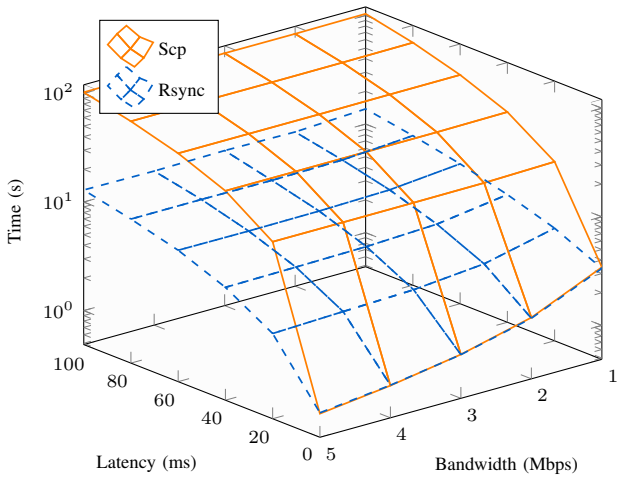

Figure 8. Performance of `scp` and `rsync` performance for different values of latency and bandwidth.

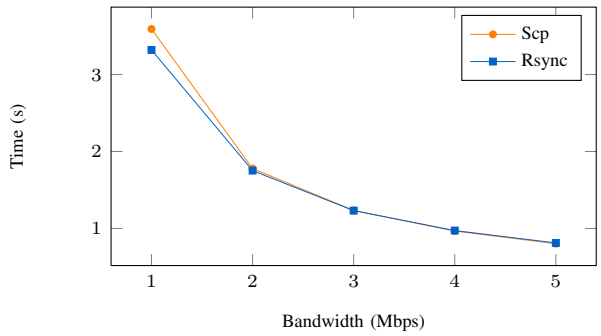

Figure 9. Comparison between `scp` and `rsync` (bandwidth).

Performance Linpack, DGEMM, and FFT benchmarks in a single virtual nodes with 1 or 4 cores, and we compare the GFlops obtained when varying the emulated frequency. We also varied the emulation algorithm to use CPU-Hogs and CPU-Gov. Each experiment has been executed 20 times and averaged.

The results are shown in Figures 11 and 12. The frequencies emulated have been chosen according to the available CPU frequency steps. Thus, using CPU-Gov is like modifying directly the CPU frequency with the Linux features (*cpufreq*).

Even if it is not perfect in all the situations, CPU-Hogs follows the same shape than a true hardware limitation.

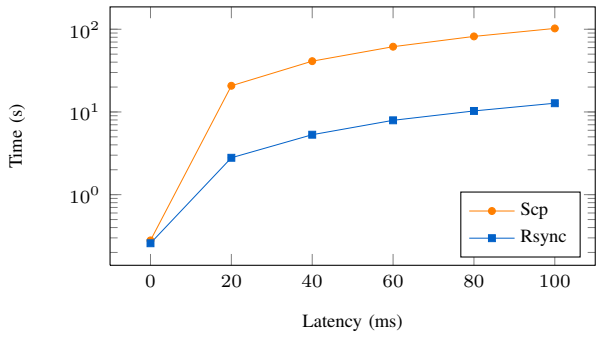

Figure 10. Comparison between `scp` and `rsync` (latency).

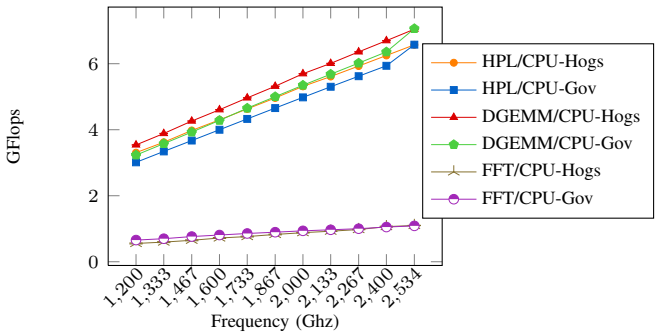

Figure 11. HPL, DGEMM and FFT benchmarks with 1 CPU.

Furthermore, this Distem feature is really interesting to emulate frequencies that are not natively supported by processors (e.g. processors with poor *cpufreq* support). A more complete study on CPU emulation can be found in [3].

C. Large scale deployment

Distem is able to run multiple virtual nodes inside one physical node. This gives an opportunity to create large, virtual configurations with only limited amount of computing hardware. In this section we show that Distem scales well with the number of virtual nodes and is able to emulate large computing infrastructure with modest means.

Infrastructure deployment: In this first experiment, we measured the time to get a fully working virtual infrastructure and to execute a parallel command on all the deployed virtual nodes. We measured the time required:

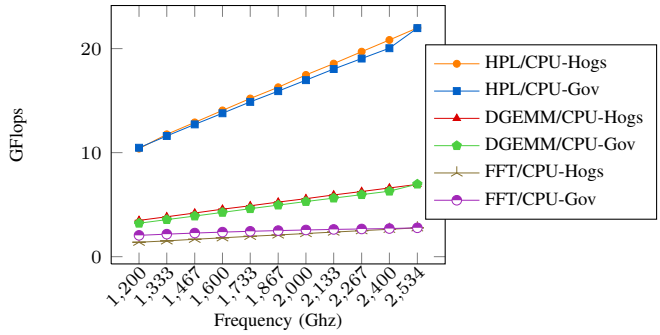

Figure 12. HPL, DGEMM, and FFT benchmarks with 4 CPU.

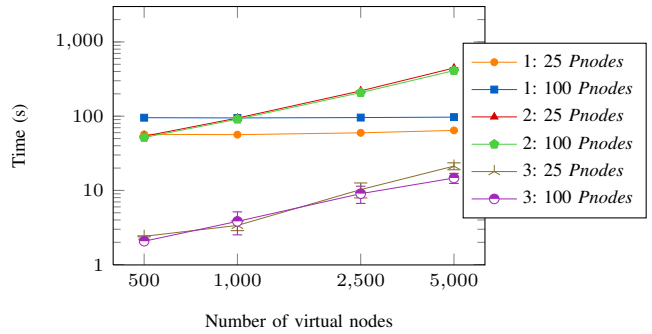

Figure 13. Large scale virtual infrastructure deployed on 25 and 100 *Pnodes*. 1 means "*Pnodes* installation", 2 means "*Vnodes* deployment", and 3 means "TakTuk execution".

- to install Distem on all the *Pnodes* (packages installation and network configuration);
- to launch the *Vnodes*;
- to execute a command on all the virtual nodes with TakTuk [5].

The measures have been taken with:

- 25 and 100 *Pnodes*;
- 500, 1000, 2500, and 5000 *Vnodes*.

Each experiment has been executed ten times, then averaged. The results are shown in Figure 13. We can see that the time to launch the virtual nodes does not really depend on the number of physical nodes. Deploying a 5000 nodes virtual infrastructure takes less than ten minutes. We can also observe that the TakTuk execution time decreases when the density of virtual nodes decreases, this is a normal behavior because physical nodes have an increased response time due to a lower load. Thus, depending on the experiment that must be performed, the virtual node density has to be taken into account more or less carefully.

Hierarchical parallel commands tool: Here, we deployed 2560 nodes on 10 nodes.The goal was to study the communications of the TakTuk [5] parallel command runner. TakTuk uses hierachical communications to reach all the nodes and the communication tree is built using an adaptive algorithm.

Figure 14 shows the deployment tree used by TakTuk to distribute communication.

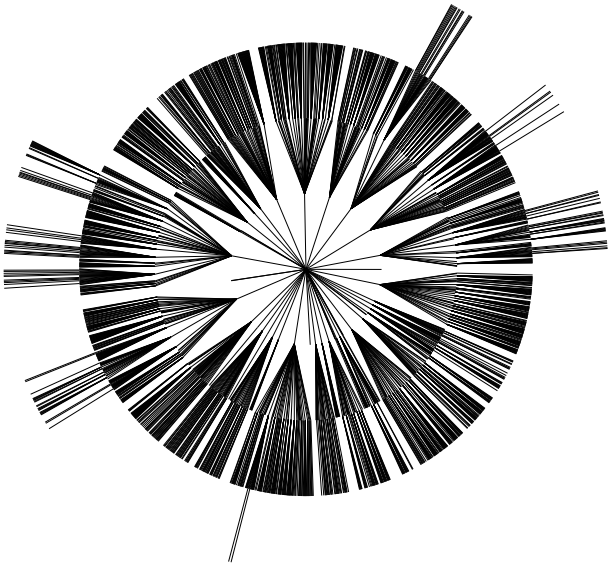

Figure 14. A radial tree representation of the TakTuk communication topology with 2560 nodes. The node initiating the request is in the centre.

IV. Related work

There exist a large number of work aiming at building experimental environments providing controlled performance of CPU and/or network inside virtual network topologies. However, many of them are not publicly available, or are no longer maintained, which limits the possibilities of comparisons to the descriptions available in publications. In order to mitigate this issue with Distem, we limited our choice of building blocks to those available in the standard Linux kernel, and released the source code as well as documentation under a free software license. We have successfully verified that third-party users were able to perform experiments using Distem.

Regarding network emulation, most related works rely on network link emulators [4] such as Dummynet [6] (used e.g. by Emulab [7], PlanetLab [8] and P2PLab [9]), NIST-Net [10] or Linux TC (used by Wrekavoc [11] and Distem). Those tools are limited to single-link network emulation, but are generally used together with topology descriptions to create emulated network topologies running on a set of nodes.

Solutions differentiate on the kind of network topologies that are emulated, from more simple and scalable (eWAN [12], P2PLab [9], Wrekavoc [11]) to more realistic and detailed (Emulab [7], Distem). There also exist more integrated approaches like ModelNet [13], where a single tool handle the emulation of network topologies without relying on a more low-level tool. Those software solutions have different level of availability and usability. Dummynet

is actively maintained as a component of FreeBSD, and it is also available as an external set of modules for Linux and Windows. Modelnet is no longer actively maintained, and setting it up requires using older versions of FreeBSD, which can raise hardware support issues. NISTNet is no longer maintained. Linux TC is actively maintained and part of Linux.

Some solutions use no virtualization or CPU performance emulation. In that case, the user is limited to the performance of the real machines. Another approach consists in slowing down the perception of the time for the virtualized nodes, making it possible to run more nodes on one physical node [14], [15]. The experiment will take more time to finish, but its scale can be proportionally increased. However, this approach is rather complex, and special care must be taken with devices, because they will seem to operate faster as well. This is both an advantage (faster networks and CPUs can be emulated) and a disadvantage (if more control over the device speed is needed). The approach used in Distem (based on [3]) is more simple and more robust, but is limited to reducing the performance – it cannot be used to emulate faster machines.

V. Conclusions and Future Work

In this paper, we presented Distem, a software tool to build distributed virtual experimental environments. Using an homogenenous set of nodes, Distem emulates a platform composed of heterogeneous nodes (in terms of number and performance of CPU cores) managed using Linux Containers, connected to a virtual network described using a realistic topology model, and emulated using Linux TC/netem. Through experimental validation, Distem is shown to be both accurate and scalable.

In the future, we will continue to work on improving Distem's efficiency and scalability in order to reach experiments with 100000s of nodes. In terms of features, we plan to extend Distem with the ability to emulate churn and faults, both on the machine and on the network level.

Acknowledgments

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr).

References

[1] J. Gustedt, E. Jeannot, and M. Quinson, "Experimental Methodologies for Large-Scale Systems: a Survey," *Parallel Processing Letters*, vol. 19, no. 3, pp. 399–418, 2009. [Online]. Available: http://hal.inria.fr/inria-00364180

[2] R. T. Fielding and R. N. Taylor, "Principled design of the modern Web architecture," *ACM Trans. Internet Technol.*, vol. 2, pp. 115–150, May 2002. [Online]. Available: http://doi.acm.org/10.1145/514183.514185

[3] T. Buchert, L. Nussbaum, and J. Gustedt, "Methods for Emulation of Multi-Core CPU Performance," in *HPCC-2011*, 2011.

[4] L. Nussbaum and O. Richard, "A Comparative Study of Network Link Emulators," in *CNS'09*, 2009.

[5] B. Claudel, G. Huard, and O. Richard, "TakTuk, adaptive deployment of remote executions," in *HPDC'09*, 2009.

[6] M. Carbone and L. Rizzo, "Dummynet revisited," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 2, pp. 12–20, Apr. 2010. [Online]. Available: http://dx.doi.org/10.1145/1764873.1764876

[7] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An Integrated Experimental Environment for Distributed Systems and Networks," in *OSDI'05*, 2002.

[8] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak, "Operating system support for planetary-scale network services," in *NSDI'04*, 2004.

[9] L. Nussbaum and O. Richard, "Lightweight Emulation to Study Peer-to-Peer Systems," in *Hot-P2P 06*, 2006.

[10] M. Carson and D. Santay, "Nist net: a linux-based network emulation tool," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 3, pp. 111–126, Jul. 2003. [Online]. Available: http://doi.acm.org/10.1145/956993.957007

[11] L.-C. Canon and E. Jeannot, "Wrekavoc: a Tool for Emulating Heterogeneity," in *HCW'06*, 2006.

[12] P. Vicat-Blanc Primet, O. Glück, C. Otal, and F. Echantillac, "Emulation of a grid network cloud: eWAN," INRIA Research Report RR-5449, 2004. [Online]. Available: http://hal.inria.fr/inria-00070558

[13] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostić, J. Chase, and D. Becker, "Scalability and Accuracy in a Large-Scale Network Emulator," *SIGCOMM Computer Communication Review*, vol. 36, pp. 271–284, December 2002. [Online]. Available: http://doi.acm.org/10.1145/844128.844154

[14] D. Gupta, K. V. Vishwanath, and A. Vahdat, "DieCast: Testing Distributed Systems with an Accurate Scale Model," in *NSDI'08*, 2008.

[15] Y. Zheng and D. M. Nicol, "A Virtual Time System for OpenVZ-Based Network Emulations," in *PADS'11*, 2011.