VIRCONEL: A NEW EMULATION ENVIRONMENT FOR EXPERIMENTS WITH NETWORKED IT SYSTEMS

Yacine Benchaïb and Artur Hecker
Department of Computer Science and Networking
TELECOM ParisTech (ENST)
37-39, rue Dareau, 75014 Paris, France
E-mail: {benchaib, hecker}@enst.fr

KEYWORDS
Modeling, Simulation and Evaluation Techniques ; Grid and Cluster Computing.

ABSTRACT

In this paper we present VIRCONEL, a new, open-source emulation environment for experiments with and evaluation of networked IT systems. Based on previous open-source projects, VIRCONEL proposes a graphical modeling interface with node template support, entity cloning, IP configuration auto-completion and an easy scenario definition with label-based multiple role assignment and local script execution on virtual machines. Moreover, VIRCONEL has a graphical interface for the control of the deployed virtual network, allowing in particular one click logins, monitoring and value recording, as well as link and node fault injection. Most importantly, VIRCONEL easily installs on typical PC hardware and features explicit support for multiple physical hosts, thus providing a better scaling. Multiple physical hosts are seamlessly supported both in the virtual network design and operation phases.

INTRODUCTION

The evaluation of large, networked IT systems often raises questions with regard to the best evaluation environment. This is a known issue in the evaluation of performance, robustness and assurance properties of distributed systems and applications, new distributed maintenance algorithms, middleware architectures, P2P proposals, etc. (Jiang and Xu 2003). The problem is that formal approaches are either very difficult to apply or need certain assumptions that are difficult to verify in practice. The alternative is the experimental evaluation. Real testbeds are attractive because they are often more representative than other experimental evaluations. However, they inflict a high administrative burden (deployment, maintenance, operational effort). In practice, this results in prohibitive limitations of evaluable system sizes. On the other hand, classic simulation techniques can easily deal with thousands of nodes. Yet, they impose a controversial tradeoff between precision, complexity and control (Bavier et al., 2006). When left at the consideration of the author alone, this deserves doubts with respect to the trustworthiness of the results (Pawlikowski, 2002).

Emulation using virtual networks and virtual machines represents an interesting alternative to the experimental evaluation techniques (Ruth et al. 2005). Note that in this paper, we do not distinguish between different virtualization technologies and use the terms virtualization and emulation in their broad sense. See (Nanda and Chiueh, 2005) for background details.
Essentially using the same software as real testbeds but in virtual execution environments, emulation is very close to real, at least regarding local node behavior. Differences are in the performance and capacities of virtual nodes, especially when several virtual machines share one physical host. More importantly, there can be substantial differences in the link behavior. This is essentially comparable to simulation issues: e.g. accepted models are necessary to simulate a wireless link. Still, emulation can represent an attractive alternative to real testbeds and simulations. First, emulation by virtualization features binary compatibility with the real testbed and therefore, unlike simulations, does not need an additional model programming. This also permits for closed-code execution as emulated instances, thus allowing evaluation of commercial software whose behavior might be not completely known. Besides, since model programming is not necessary, and the software to be evaluated can be used directly, this avoids a potential error or imprecision source, thus yielding results closer to real than the simulation. Second, emulation can be used to evaluate networked IT environments composed of several hundreds of nodes with a relatively low deployment and operations effort in comparison to a full testbed, and this in a fully controlled environment. However, to do this, we need to supply the evaluator with a toolbox permitting to easily set up and control different virtual environments spanning over several real hosts.
In this paper, we present the design and implementation of Virtual Computer Network Lab (VIRCONEL), an open-source, easy-to-use, multi-host, networked emulation environment for the evaluation of large, networked IT systems with the help of several off-the-shelf PCs. VIRCONEL features a very easy installation method and explicitly supports several physical hosts for better scalability. VIRCONEL features graphical interfaces for the design of the system, emulated service deployment, scenario definition and emulation control. In operation, VIRCONEL can record typical parameters and the data as requested from the emulated network.

Proceedings of the 2008 High Performance
Computing & Simulation Conference ©ECMS
Waleed W. Smari (Ed.)
ISBN: 978-0-9553018-7-2 / ISBN: 978-0-9553018-6-5 (CD)

The rest of the paper is organized as follows. In the next section, we present our rationale and the resulting requirements. We then present previous work in this area and explain the motivation for the design and development of VIRCONEL. Then, we explain its architecture and justify some of our decisions by providing insights on the related development effort. Next, we present the possibilities already provided for emulated system design, control and measurement. After that, we demonstrate the resource usage of VIRCONEL when running typical scenarii on our hardware and try to estimate its limits. Finally, we give an outlook to our future work.

RATIONALE AND REQUIREMENTS

The main drive for this work comes from the need for the setup and evaluation of large, distributed IT systems within the scope of the ICT FP6 DESEREC project. The DESEREC testbed needs to be capable of hosting different types of networked enterprise IT systems, often running complex, commercial closed-source software. Typical services within such systems include VoIP sessions between different locations, Web-based access to SQL/LDAP databases etc., provided over several LANs connected by routers over VPNs and the Internet and completed by obligatory security, reliability and management subsystems.

In practice, this translates to very close to real testbed of potentially several hundreds of nodes and a strong accent on the local applications and node behavior, i.e. being capable of running common open and commercial software under close-to-real constraints (Unix-like OS, TCP/IP plus NAT+DHCP, firewalls, NAS and AAA, etc.). Besides, to evaluate system behavior and resilience faced with node and communication breakdowns, we needed a possibility for a scenario generation, including failure scenarios. Finally, to allow collaborative partner work, the ease of installation of the emulation environment per se and the support for the interconnection of several local environments are considered important.

In summary, we distilled our wish list for an emulation package for large IT system evaluation to the following concrete MUST requirements:

-- *Open-source emulation software:* the emulation toolbox itself should not involve complicated licensing issues and be based on open-source software;

-- *Relative ease of installation:* complex installations on real hosts would be unattractive since the goal is to install the testbed and not to maintain the physical host;

-- *Support of several physical hosts:* one physical host usually cannot run more than several dozens of virtual machines. To support scaling to several hundreds of emulated nodes, we therefore need a possibility to easily support multiple, networked physical hosts. This support also needs to be integrated with the modeling of the network, i.e. it should be possible to assign virtual nodes to physical hosts;

-- *Ease of scenario definition:* the modeling of the emulated system and services upon it should be easy, preferably supported by an easy-to-use graphical tool. It should be possible to start the defined emulated network upon the available physical hosts;

-- *Binary compatibility with the existing software:* it should be possible to evaluate the existing software without understanding how it works (Ruth et al., 2005). As explained before, this has a double advantage of permitting direct usage of the existing software, e.g. closed source. What is more, it significantly reduces the modeling time, since one does not need to model local node behavior. This removes a potential error source.

-- *Running emulation monitoring and control:* it should be possible to influence a running emulation by provoking node and communication breakdowns, starting and stopping software on virtual machines, reconfiguring interfaces, etc. On the other hand, it should be possible to see what is happening within the emulation and, in particular, capture and record values of different interesting variables, limiting the perturbation of the emulated virtual reality.

In principle, our list corresponds to the requirements stated in (Bavier et al., 2006), which mainly underlines realism (real software, realistic conditions, real traffic) and controllability.

RELATED WORK

Network experiments are conducted today mainly through simulations with NS-2, OMNET++, Glomosim or commercial tools like Opnet. As discussed above, without a substantial additional effort, the simulation tools do not allow direct execution of closed-source, (e.g. commercial) software.

PL-VINI (Bavier, 2006) running over PlanetLab has been proposed for similar purposes, but access to PlanetLab is not always suitable.

Different “local” virtualization environments with networking support have been proposed, including commercial environments like VMWare and Parallels, and open-source projects like Qemu (Bellard, 2005), openVZ (http://openvz.org/), Xen (Barham, 2003) and User-Mode Linux (UML) (Dike, 2006) . Besides the previously cited survey (Nanda and Chiueh, 2005), an up-to-date comparison of these can be found in Wikipedia under “Comparison_of_virtual_machines”.

A substantial work has been done on server virtualization and containment (see e.g. Padala, 2007). However, the aim of this work is on the one hand on the performance and high availability of any single server, and on the hand, on the management of such servers. We would like to place as many virtual or paravirtual instances on any single physical host so as to make our network emulation scale. We also need the control, but focus on scenario control and monitoring, rather than on the service management (patches, security updates, user/account management, etc.), typical for real servers. Besides, we need a tool to graphically define virtual topologies.

Several open-source projects add virtual networking support to virtual machines. For instance, Netkit (http://www.netkit.org/) is a collection of shell scripts

for instantiating a virtual network of UML-based virtual machines. VN-UML (Galan, 2004) and MLN (http://mln.sourceforge.net/) support structured descriptions of the network to be set up on one machine, with MLN also supporting Xen and UML combinations. Graphical editors emerged for such structured descriptions, like NetGUI (Nemesio, 2006), vnumlgui (http://pagesperso.erasme.org/michel/vnumlgui/) producing VN-UML's XML. Marionnet (Loddo, Saiu 2007), principally accentuating on didactics and dedicated to teaching, adds dynamic network reconfiguration support.

Having studied the related work, we concluded that very interesting building blocks exist in the open-source community. On the other hand, no proposal permitted to fulfill all of our requirements. Especially the graphical editors building upon VN-UML generally come close to our requirements. However, all of them are limited to one physical host, both in the modeling and in the emulation execution phases. Second important point (Bavier et al., 2006): they generally do not integrate monitoring and control utilities. Marionnet features support for topology changes in operation. However, it targets education purposes and is rather committed to realism when working with small networks, while we would like to simplify modeling work.

DESIGN AND IMPLEMENTATION OF VIRCONEL

Profiting from previous experiences and trying to reduce precious development time, VIRCONEL relies upon and extends VN-UML (Galan, 2004).

VN-UML uses UML as virtual machines. UML is in principle a Linux kernel started in the user-space, as any other process. It can thus be stopped and interrupted at any time. For networking support, the VMs make use of the Linux kernel's ability to provide virtual network interfaces (*tun* and *tap* devices). The interface of the UML VM (*eth0*) connects to such a virtual interface of the host Linux. By defining proper IP forwarding and/or bridging rules, any VM can get customized network access. To simplify the necessary configuration, VN-UML introduces a virtual switch and a structured definition language, which defines the interconnects of VMs with each other and the physical host. Isolation is possible through the use of VLANs (see Figure 1, PC1).

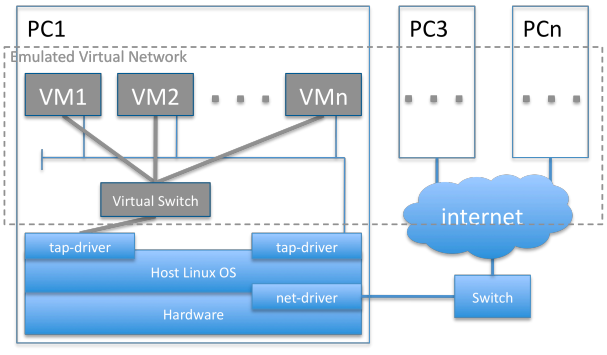

Figure 1: System architecture of VIRCONEL

In VIRCONEL, the VN-UML virtual switch is currently bridged to the real network device of the host Linux in a VLAN. If available physical hosts are interconnected by a real network (e.g. switch, router, VPN gateway, the Internet), the virtual network can span over these hosts, as shown in Figure 1. Currently this is done with limitations on topology but in principle, different isolated virtual networks can be set up with known real network separation measures (e.g. VLAN or VPN). However, this is not an urgent requirement for us.

Besides, any VM can be configured with several interfaces. An interesting point in VN-UML is the explicit presence of an additional interface, used for direct communications with the hosting physical machine. We call this interface "management interface". In VIRCONEL, we use it for operational control, scenario deployment, measurement traffic, etc.

The emulated machines are Linux kernels working over a specific file on a host PC as a shared partition. By installing software in this "partition" and preparing different partition files, we provide VM templates. The software installed within the latter can also be started and controlled over the management interface.

In principle, to run an emulation, four phases are necessary in VIRCONEL after it has been installed:

-- virtual network modeling,
-- virtual scenario definition and setup,
-- deployment of virtual entities to physical hosts, and
-- virtual network operations.

For modeling, we use a popular open-source graphical editor called *Dia* that produces XML output (http://www.gnome.org/projects/dia/). We integrated Dia by adding a VIRCONEL-specific workbench, permitting the choice of different typical entities (switches, routers, hosts). VIRCONEL's parser processes Dia's XML output and translates it into a VN-UML XML input file, producing one XML file per specified physical host as mentioned in the graphical model. Scenario setup is described in the next section.

In the deployment phase, these files are then distributed to and executed on the available physical hosts, as specified in the model file. We use SSH for that.

In the operational phase, VIRCONEL starts a control panel that displays the emulated network within one graphical interface (developed in Tcl/Tk), permitting scenario start, local login to every virtual machine, activating/deacting links, etc.

This design and the reuse of the previous work have permitted us to accomplish a first working version of VIRCONEL in about five men-months of integration work. In the following, we describe the usage and features of VIRCONEL.

USAGE AND FEATURES OF VIRCONEL

Local Installation

Similarly to VN-UML, we use the Live-DVD concept permitting a very easy local deployment on a spare physical host. This allows concentrating on the essential, emulation-related things.

However, since UML, unlike other virtualization technologies, does not necessarily require host machine changes, VIRCONEL can also be installed and executed on an available Linux host used for other tasks.

Modeling: Virtual Topology Definition

Modeling is done within the integrated graphical editor (Dia, slightly modified). Currently, VIRCONEL comes with switch, router and host templates. The existing templates, available as icons in Dia, can be positioned on the screen and interconnected by links as necessary. Graphical links represent emulated network links.
The provided host template comes with a variety of typical applications, including Web server, client, SIP instances, etc. It is possible to change the existing/to add new VM templates at any time (and to integrate them into the graphical tool).
To further simplify things, we explicitly support starting and stopping processes on any operational VM from the modeling phase on. This is used for assigning roles for the scenario definition (see below), but also renders any usage of the existing templates more flexible, since the same template can be used to instantiate semantically different VMs (e.g. a server and a client).

Setup: Scenario Definition

To assign such roles and/or configuration parameters, we use a simple *labeling* technique. Designer can attach a number of text labels to any existing basic entity. When all labels are assigned, the designer simply groups all labels with the original entity using Dia's grouping function. This attaches the labels to this specific entity. Therefore, designer can define IP addresses, specify which processes should be started, etc. We also provide some support for rapid modeling, namely *IP-configuration auto-completion* and simple entity *cloning*. The auto-completion function can automatically find the responsible router from the XML topology file. Hence, attaching an IPv4 address in CIDR to each host is sufficient. Cloning is very useful to produce high numbers of identical hosts. Currently, it is possible to attach a <clone=N> label to a well-defined host in order to clone the latter N times (thus resulting in N+1 identical entities with the same behavior). The IP addresses of the emulated interfaces are automatically renumbered within the subnetwork space.
The scenario per se is defined through labels, which identify scripts to be executed on each concerned modeled entity. More precisely, the <*-Client> and <*-Server> labels are interpreted as parameters to a launcher script. The latter searches and executes the script with the same name within the targeted VM. Thus, the scenario definition needs a machine pre-provisioning with all required executables (template, or copying by hand in the operational machine), various script placements on the machine (with names corresponding to the labels) and the definitions of resource consumption models (e.g. period, number of bytes to send, etc.) within the scripts.
Such scripts can be developed by the designer and are very flexible (basically, shell script, perl, python etc.). They permit to define any typical scenario, for instance a number of Web clients accessing a multi-tiered Web server with certain distributions, etc. (Padala, 2007). Since any script and binary execution is supported on virtual machines, this approach does not constrain the possibilities. A list of currently supported labels with their semantical meaning is given in Table 1 but is being constantly worked on.

Table 1 Currently supported VIRCONEL labels

Label	
*-Client	Used to start a client script on the VM
*-Server	Used to start a server script on the VM
Clone	Clone a specified virtual machine.
IP address	Define the VM's IP address (IP/mask)

Deployment

We support multiple physical hosts in modeling, setup and execution phases. The designer needs to assign virtual machines to physical servers. In VIRCONEL, this can be done by enclosing a required number of subnets/hosts and a router into a graphical rectangle in Dia. The designer then specifies the IP address of the physical host as shown in Figure 2. Once these phases are accomplished, the emulation can be started by parsing the produced output file. This locally starts the operational GUI, which exactly represents the whole modeled topology hiding the physical hosts as can be seen in Figure 3. It permits to start/stop both the virtualization and the defined scenario and has some other features to be described in the next phase.
Driven over this GUI, which uses SSH from the designer host to physical hosts, VIRCONEL first distributes the designed virtual topology within the specified testbed and then initiates the virtual network entities necessary to combine the subnetworks hosted on different physical machines. The testbed is composed of PCs, each of which is running VIRCONEL. The virtual machines are started on the physical hosts of the emulation platform as identified by their IP addresses.

Operation

Furthermore, the same GUI also permits to control the operation of the emulation. It is possible to launch and stop the defined scenario. Commands are sent over SSH from the designer host to each VM as specified.
Second, the operation GUI features a *one-click-login* to any virtual machine, which opens an SSH session from the designer host to the virtual machine's management interface. This is very practical for manual error introduction or for tests/measures and slight changes within the operational virtual environment.

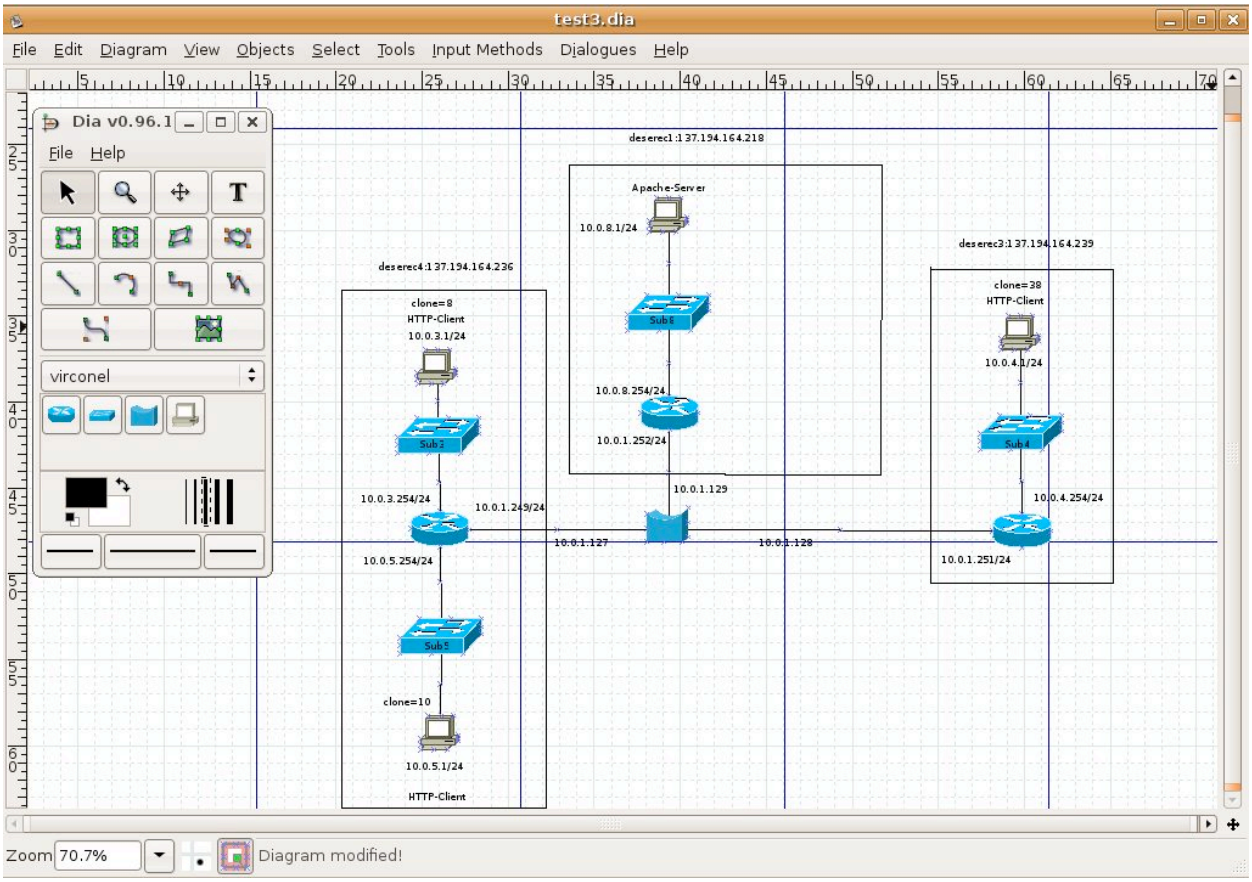

Figure 2: VIRCONEL modeling interface

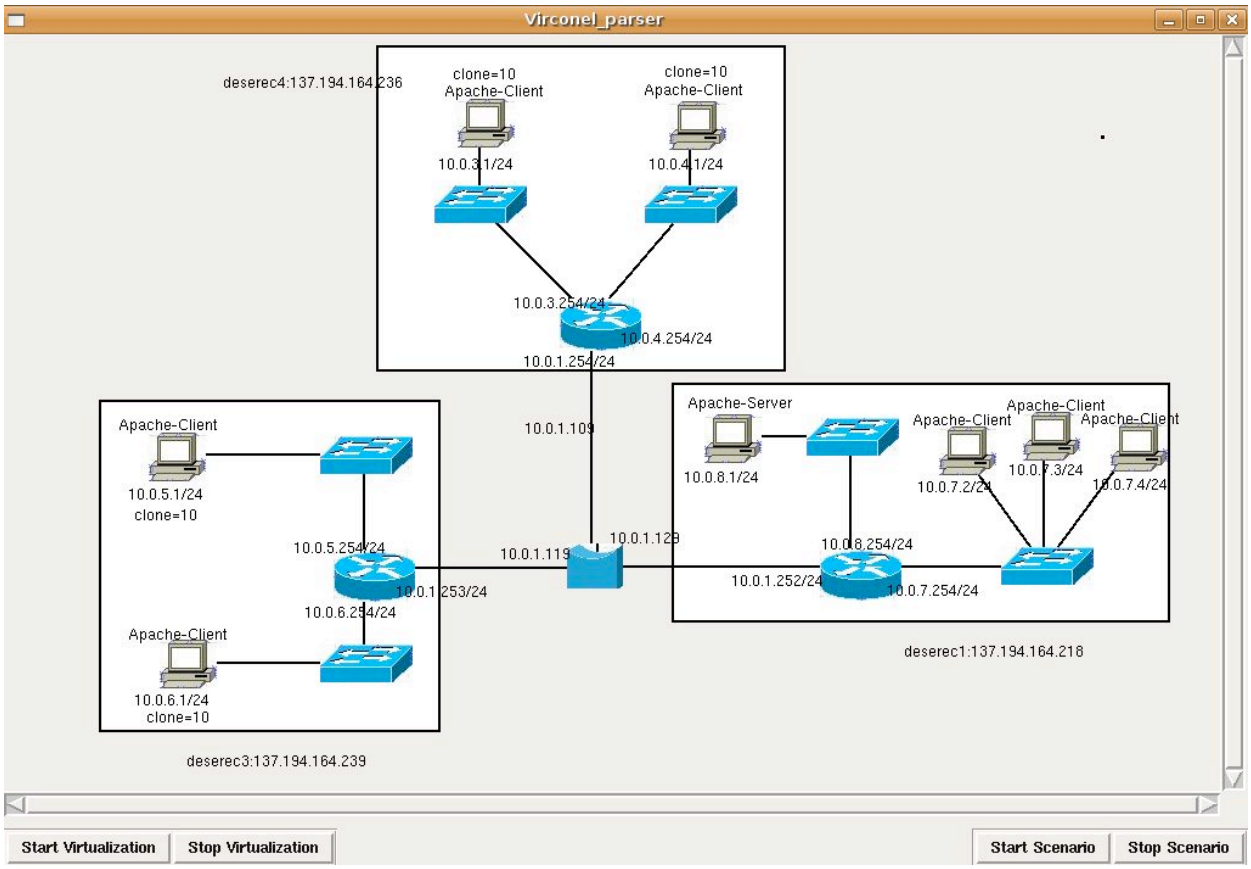

Figure 3: Control of the emulated network

The management interface permits to collect various measurements without perturbing the emulated network traffic. Per default, VIRCONEL assesses typical data, like overall CPU consumption and overall network traffic on every emulated interface, and represent these with *gnuplot*. Yet, more complicated measurements can be defined in the setup phase. In principle, whatever can be measured in the real network can be measured in VIRCONEL. The assessed data is either sent over the management interface to the operational GUI, or it is stored in the virtual or physical host partition. Resource usage measurements are also possible from the host PC.

EVALUATION RESULTS

Evaluation Testbed

The testbed on which we install VIRCONEL and run our evalution is composed of three servers (PC), each equipped with 4GB of RAM and a 2.6GHz QuadCore Intel CPU. The servers are running Ubuntu Linux 7.04 Feisty Fawn, kernel version 2.6.20 patched with the SKAS3 patch for better UML performance.

Evaluated Scenarios

We use three scenarios to evaluate VIRCONEL. The first scenario evaluates the computational penalty experienced by a process within the virtual machine. We want to find answers as to how much performance we lose per VM when running several VMs on the same host. This gives an estimate on how many concurrent VMs we can put on one Linux host.
We use *openssl* to symmetrically encrypt a 20MB binary file. We first sample the host Linux performance and then repeat the exact same command within the VM with concurrent 6, 11 and 16 VMs on the same host Linux. On our hardware, the host Linux performs this task (measured with *time*) in an average time of 1.048s with a standard deviation of 0.114 s. The VMs take an average of 2.5s (0.447) for 6, 3.265s (0.432) for 11 and 6.702s (1.381) for 16 concurrent VMs on one physical host respectively (standard deviation in brackets). The results of these measurements as percentage, normalized to fixed host performance, are shown in Figure 4.

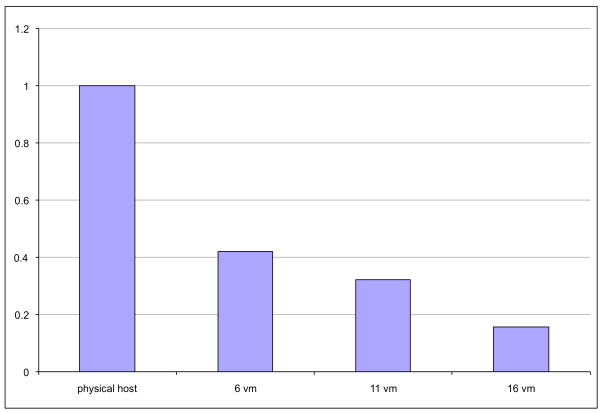

Figure 4: Computation performance penalty per VM on one host PC with the increasing number of VMs

We can see that at least up to 11 VMs can be used for this computationally intense task with a reasonable and stable penalty. For 16 VMs the results start varying too much because of complex interactions of concurrent processes with the task scheduling. Note that while the performance per VM decreases, the overall performance for at least up to 16 VMs is better than for the single process at the host Linux: while the host Linux takes 1.0457s per file encryption, 6 VMs take only 0.42s per file, 11 VMs take 0.296s per file and 16 VMs take 0.42s per file. We can see that 11 concurrent VMs have the best performance in that scenario.
In the second scenario, we use three physical hosts connected by a real switch (100BaseT). We emulate HTTP traffic from virtual clients (*wget*) to one virtual webserver (Apache). This roughly represents a mixed resource usage typical for a modern distributed application. Using Unix *at*, HTTP client starts on all client VMs simultaneously, sends an HTTP request to the webserver for a hosted file of 50kB and exits immediately. We use the topology as illustrated in Figure 2: we use 3 host Linux PCs, with the webserver and router being the only VMs on the host deserec2. There are 11 concurrent HTTP clients on deserec1 (plus router VM) and 16 HTTP clients on deserec4 (idem). We measure the delay for a succesfull transaction from within the VM, i.e. the time from the start to the exit of *wget*. The averaged results for physically identitical hosts deserec1 (11 clients) and deserec4 (16 clients) in 20 experiments are illustrated in Figure 5.

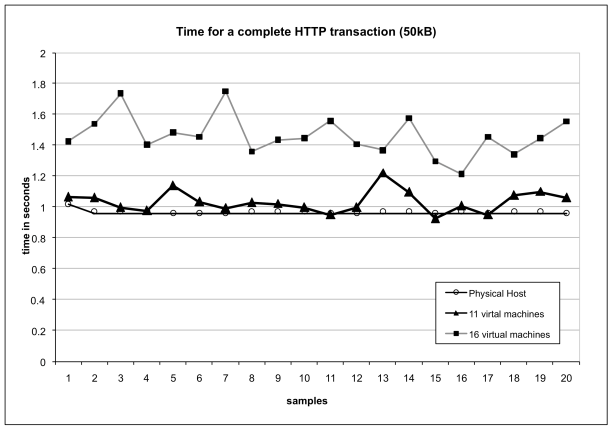

Figure 5: Time for a complete execution of an emulated HTTP transaction with 11 and 16 concurrent VMs

Third scenario is similar to the second one. We use HTTP traffic from virtual clients to one virtual server in the same topology (Figure 2). However, we start the clients periodically, with the inter-process invocation time from a uniform distribution in the interval [1s..2s], independent for every VM. The client starts, sends an HTTP request to the Web server for a hosted file of 50kB and exits. The concurrent process start/end produces a considerable I/O activity.
Under these conditions, we vary the overall number of client VMs on deserec1 and deserec4 host PCs and measure the CPU consumption on the physical host. The limit is reached for 27 VMs due to the frequent

process starts and stops on the concurrent VMs. In Figure 6, we show CPU utilization on the host Linux under the number of concurrent VMs. For scenario 3 the increase is linear. Hence, in similar scenarios, it is possible to maintain a ratio of about 20 VMs per physical host.

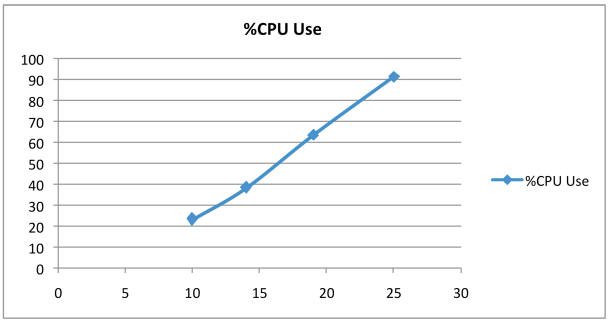

Figure 6: CPU usage on one physical host (Y axis) with different number of UML virtual machines (X axis)

Current VIRCONEL Limits

The usage of VN-UML and of UML technology imply several limits. First, VIRCONEL is a Linux-only environment, both for physical and virtual machines. Second, if with VIRCONEL it is possible to use several physical hosts, in practice this will reach management limits. Also, the assignment of VMs to physical hosts is done manually. However, the main limitation is within the topological constraints: it is currently necessary to specify a virtual router per physical host. For our work in DESEREC, this is not a serious problem. But we may consider this point in our further work.

CONCLUSION

VIRCONEL is a very easy to install and rather simple to use emulation environment for experiments with IT systems. Compared to the existing work, our main contributions are the intrinsic support for multiple physical hosts and integrated monitoring and control. With VIRCONEL, it is interesting to combine real and emulated entities. In that manner, resource-demanding entities can be treated separately, while numerous small entities can be easily cloned in the emulation.
In future we plan to improve the functionalities of VIRCONEL, namely its monitoring capabilities, tying these to the modeling. VIRCONEL can be freely (GPL) downloaded from http://www.infres.enst.fr/~deserec.

ACKNOWLEDGMENTS

This work is supported in part by the EC ICT FP6 DESEREC (CN 026600, www.deserec.eu) project. The authors are thankful to the anonymous reviewers whose valuable remarks helped to improve this work.

REFERENCES

Barham P.; Dragovic, B.; Fraser, K.;, Hand, S.; Harris, T.; Ho, A.; Neugebauer, R.; Pratt, I.; Warfield A. “Xen and the Art of Virtualization”, in proc. ACM HotNets-I, ACM Press, 2003, pp. 59-64.

Bavier, A; Feamster, N.; Huang, M.; Peterson, L.; and Rexford, J. “In VINI Veritas: Realistic and Controlled Network Experimentation”, in proc. ACM SIGCOMM 2006, Pisa, Italy.

Bellard, F. “Qemu, a Fast and Portable Dynamic Translator”, in proc. FREENIX Track of USENIX 2005 Annual Technical Conference, pp. 41-46.

Dike, J. “User Mode Linux”, Prentice-Hall, April 2006.

Galan, F.; Fernandez, D.; Ruiz, J.; Walid, O.; de Miguel, T. “Use of Virtualization Tools in Computer Network Laboratories”, in proc. 5th IEEE ITHET, June 2004.

Jiang, X.; Xu, D.. “vBET: a VM-Based Emulation Testbed”, in proc. ACM SIGCOMM 2003, Karlsruhe, Germany.

Loddo, J.-V.; Saiu, L. “Status Report: Marionnet. How to implement a Virtual Network Laboratory in Six Months and Be Happy”, in proc. ACM ML’07, Freiburg, Germany, October, 2007.

Nanda, S.; Chiueh, T.. “A Survey on Virtualization Technologies,” the Research Proficiency Report, Stony Brook, ECSL-TR-179, February 2005.

Nemesio, S. C.; de las Heras Quiros, P.; Barbero, E. M. C.; Gonzalez, J. C. “Early Experiences with NetGUI Laboratories”, SIIE’06, Leon, Spain, October 2006.

Padala, P.; Zhu, X.; Wang, Z.; Singhal S.; Shin, K. G. “Performance Evaluation of Virtualization Technologies for Server Consolidation”, HP Laboratories Technical Report, April 2007 (available online).

Pawlikowski K.; Jeong, H.-D. J. and Lee, J-S. R. “On Credibility of Simulation Studies Of Telecommunication Networks”, IEEE Communications Magazine, January 2002, pp. 132-139.

Ruth, Paul; Jiang, Xuxian; Xu, Dongyan; Goasguen, Sebastien. “Virtual Distributed Environments in a Shared Infrastructure”, IEEE Computer, May 2005, pp. 63-69.

AUTHOR BIOGRAPHIES

YACINE BENCHAÏB holds a Master in Computer Science from the Université d’Amiens, France. In 2007, he joined the networking and computer science department of TELECOM ParisTech (ENST), where he currently works as research engineer. His research work includes virtualization and network security. Contact him under `benchaib@enst.fr`.

ARTUR HECKER holds a diploma in Computer Science (Dipl.inform.) from the University of Karlsruhe (TH), Germany and a PhD degree in Computer Science and Networking from the ENST, France. He worked as CTO of Wavestorm SAS, which he co-founded in 2003. Since 2006, he is Associate Professor at the INFRES department at the ENST. His present research interests are wireless access security, autonomous networking and security assurance of complex systems. Dr. Hecker is actively involved in several IST FP6 and EUREKA CELTIC research activities. Contact him under `hecker@enst.fr`.