High Accuracy Failure Injection in Parallel and Distributed Systems Using Virtualization

Thomas Hérault
Univ Paris Sud; LRI; INRIA;
F-91405 Orsay France
thomas.herault@lri.fr

Thomas Largillier
Univ Paris Sud; LRI; INRIA;
F-91405 Orsay France
thomas.largillier@lri.fr

Sylvain Peyronnet
Univ Paris Sud; LRI; INRIA;
F-91405 Orsay France
sylvain.peyronnet@lri.fr

Benjamin Quétier
Univ Paris Sud; LRI; INRIA;
F-91405 Orsay France
benjamin.quetier@lri.fr

Franck Cappello
INRIA;
F-91893 Orsay France
fci@lri.fr

Mathieu Jan
CEA; LIST; F-91191 Saclay, France
mathieu.jan@cea.fr

ABSTRACT

Emulation sits between simulation and experimentation to complete the set of tools available for software designers to evaluate their software and predict behavior under conditions usually unachievable in a laboratory experiment. It consists in running the real application in an emulated environment. Thus, it behaves more realistically than a simulation, but under a controlled and reproducible environment, more suitable for behavior analysis.

In this paper, we propose an emulation platform for parallel and distributed systems where both the machines and the network are virtualized at a low level. We demonstrate that the use of virtual machines allows us to test highly accurate failure injection by "destroying" virtual machines. Failure accuracy is a criteria that demonstrates how realistic a fault is. The platform accuracy is evaluated using Pastry, a fault-tolerant distributed hash-table.

Categories and Subject Descriptors

C.2.4 [**Distributed systems**]: Distributed applications; C.4 [**Performance of systems**]: Fault tolerance

General Terms

Experimentation reliability measurement

1. INTRODUCTION

One of the most important issue for the evaluation of a parallel or distributed application is to monitor and control the experimental conditions under which this evaluation is done. This is particularly important when it comes to reproducibility and analysis of observed behavior.

In general, experimental conditions are not strictly reproducible in the real world. The approach usually taken to broaden the scope of the evaluation consists in designing simulators, under which the experimental conditions can be as diverse as necessary. The "real world" experiments can then help to validate the results given by simulators under the reproduced similar conditions.

Still, simulators can only handle a model of the application, and it is hard to validate an implementation, or guarantee that the end user application will meet the predicted performance and behavior. Here, we study another tool for experimentations: emulators. Emulators are able to run the final application, under emulated conditions.

Within an emulated environment, the experimenter can inject experimental conditions that are not accessible in a real environment, or not controlled. A typical example of such condition is the apparition of hardware failures during the experiment. With a real system, although they can happen undeterministically, hardware failures are hard to inject, and hard to reproduce. In an emulated environment, hardware is software-controlled and the experimenter can design a reproducible scenario of fault injection to stress fault tolerant applications.

A promising approach for emulators is the use of virtual machines (VM). A VM by itself fits partially the goals of parallel application emulators, since it emulates instances of a virtual hardware on a single machine. In addition to these virtual machines, we need to link them through a controllable network. In this paper, we present V-DS, a platform for the emulation of parallel and distributed systems (V-DS stands for Virtual Distributed System) through virtualization of the machines and the network.

2. RELATED WORK

Recently, the number of large-scale distributed infrastructures has grown. However, these infrastructures usually fall either into the category of production infrastructures, such as EGEE [1] or DEISA [9], or in the category of research infrastructures, such as PlanetLab [6].

For instance, PlanetLab [6] is a testbed featuring nodes connected over the Internet, and software reconfiguration for experiments. However, Planetlab lacks means to control experimental conditions. Consequently, it may be difficult to apply results obtained on PlanetLab to other environments, as pointed out by [7]. Emulab [16] is an emulation platform

[1] http://www.eu-egee.org/

CF'09, May 18–20, 2009, Ischia, Italy.

that offers large-scale virtualization and low-level network emulation. However, this project focuses only on the full reconfiguration of the network stack. Moreover, Emulab uses extended FreeBSD jails as virtual machines. Inside jails, the operating system is shared between the real machine and the virtual machine, thus killing a virtual machine is the same as killing a process, which has a significant impact on the possible behaviors, as we demonstrate in the experiment section. To the best of our knowledge, Grid'5000 [3] is the only platform that provides tools to reconfigure the full software stack between the hardware and the user on all processors, and reservation capabilities to ensure controllable network conditions during the experiments. However, much work remains to be carried out for injecting or saving, in an accurate and automatic manner, experimental conditions in order to reproduce experiments.

Software environments for enabling large-scale evaluations most closely related to ours are [2] and [10]. An example of integrated environment for performing large-scale experiments, via emulation, of P2P protocols inside a cluster can be found in [2]. This work concentrates on evaluating the overhead of the framework itself and not on demonstrating its strength. In addition, the project provides a basic and specific API suited for P2P systems only. P2PLab [10] is another environment for performing P2P experiments at large-scale in a (network) controlled environment, through the use of Dummynet [12]. However, as for the previously mentioned project [2], it relies on the operating system scheduler to run several peers per physical node, leading to CPU time unfairness. Modelnet [15] is also based on Dummynet: it uses the same scheme except that the nodes that control the network are different from the computing nodes.

3. V-DS PLATFORM DESCRIPTION

The V-DS virtualization environment is composed of two distinct components: the virtualization environment for large-scale distributed systems and a BSD-module for the low-level network virtualization.

3.1 Virtualization environment for large-scale Distributed Systems

V-DS virtualizes distributed systems entities, at both operating system and network level. This is done by providing each virtual node its proper and confined operating system and execution environment.

V-DS supports three key requirements:

Scalability: In order to provide insights on large-scale distributed systems, V-DS supports the folding of distributed systems.

Accuracy: In order to obtain accurate behavior of a large-scale distributed system several constraints on the virtual machines (VMs) are needed. Using Xen allows V-DS to ensure all these requirements (see for example [11]).

Adaptivity: the platform provides a custom and optimized view of the physical infrastructure used.

V-DS is based on the Xen [1] virtualization tool (version 3.2). Compared to other virtualization technologies it has been demonstrated that Xen offers better results (see [11]).

Figure 1 shows the general architecture of V-DS.All communications between these VM are routed to FreeBSD machines to 1) prevent them from communicating directly through the internal network if they are on the same physical machine, 2) add network topologies between VM.

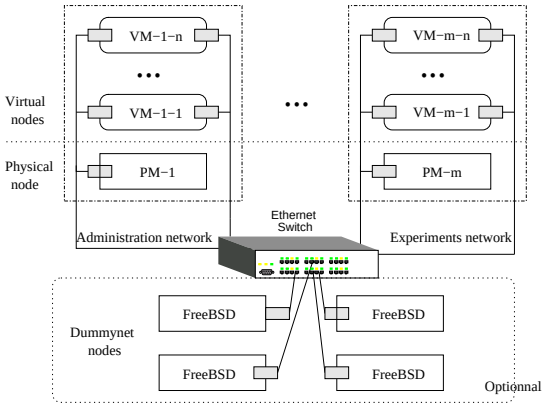

Figure 1: Overview of the architecture of V-DS.

3.2 Low-level network virtualization

One of the main advantages of the V-DS platform is that it also uses virtualization techniques for emulating the network. This allows the experimenter to emulate any kind of topology with various values for latency and bandwith on a cluster.

For the purpose of emulating the network, we use FreeBSD machines which contains several efficient tools to manipulate packages like ipfw[2] or Dummynet [12]. The platform is then capable of injecting realistic failures at the machine and network level.

There are three networks joining the virtual machines. The first is a classic ethernet network. Each virtual machine has its own virtual ethernet card. The second one uses Myrinet cards and provides very fast links between the nodes. The last one offers layer 2 virtualization using the EtherIP protocol [8]. EtherIP bridges are set between any virtual machine and the corresponding BSD machine.

The topology is given by the user in a dot format[3] file which is easy to manipulate and to write.

Using this topology file, we generate the routing table of each BSD machine. The routing is made through a kernel module. More precisely it is a netgraph[4] node called "ip_switch".

4. EXPERIMENTS

In this section we present the experiments we perform in order to assess the performances and functionalities of our virtualization framework. All these experiments were done on the Grid'5000 [3] platform. For our experiments we used a 312-node homogeneous cluster composed of AMD Opteron 248 (2.2 GHz/1MB L2 cache). Each node feature 20GB of swap and SATA hard drive. Nodes are interconnected through a Gigabit Ethernet switch. All our experiments were performed using a folding ratio of 10 (e.g. each physical node runs 10 virtual machines).

[2]http://www.freebsd.org/doc/en/books/handbook/firewalls-ipfw.html

[3]http://www.graphviz.org/doc/info/lang.html

[4]http://people.freebsd.org/~julian/netgraph.html

All the following experiments used Xen version 3.2, with Linux-2.6.18.8 for the physical and virtual computing nodes, and BSD version 7-0PRERELEASE for the network emulation.We chose the 1.5.0_10-eval version of the Java VM that fulfilled our needs and our space requirements.

4.1 Impact of the Low-Level Network Emulation

We first measured the impact of the network emulation of V-DS on the network bandwidth and latency, using the netperf tool [14]. We used two versions of V-DS: with network emulation at high level only, and with low-level network emulation, as described in section 3.

The experimental setup consisted in three physical machines: one running the BSD router, the other two running one virtual machine each. We configured the BSD router according to the restraints on the network we wanted to study.

Regarding the latency, we obtained in both cases the requested values (from 10ms to 500ms).

For the bandwith restraints, the requested values range between 256Kbps and 250Mbps. The values we obtained without the low-level network emulation are very close to the requested ones since the lost is around 3%. Another 3% is lost when adding the low-level network emulation.

4.2 Stress of Fault-Tolerant Applications

In order to evaluate the platform capabilities to inject failures, we stressed FreePastry which is an open-source Java implementation of Pastry [13, 4] intended for deployment over the Internet. Pastry is a fault-tolerant peer-to-peer protocol implementing distributed hash-tables. In Pastry every node has a unique identifier which is 128 bits long.This identifier is used to position the node on a 2^{128}-places oriented ring.

When a node is joining an existing ring, it gets a node Id and initializes its leaf set and routing table by finding a "close" node according to a proximity metric. Then it asks this node to route a special message with its identifier as a key. The node at the end of the path is the one with the closest identifier and then the new node takes its leaf set and is routing table is updated with information gathered along the path. The new node will then send messages in the pastry network to update the routing table of all processes it should be connected to.

Pastry manages nodes failures as nodes departures without notification. In order to handle this kind of situation "neighbors" (nodes which are in each others leaf set) exchange keepalive messages. If a node is still not responding after a period T, it is declared failed and everyone in its leaf set is informed. The routing table of all processes that the departing process was connected to are then updated. At some point, the routes stop changing (they are stabilized), but the maintaining procedures for these routes continue to execute.

To validate the platform we proceeded as follows. First we evaluated the average time for the system to stabilize itself after all the peers had joined the network. Then we evaluated the average time needed for every node to know that a node was shut down or killed. In the first case we only kill a java process and in the second we "destroy" the virtual machine which is hosting the process.

The experiments go as follows. The first virtual machine (called the bootstrap node) creates a new ring and then every other virtual machine connects to it. We ask every node for its routing table every 200ms and log it whenever it changes together with a time stamp.

In order not to overwhelm the bootstrap node, we launch machines by groups of tens separated by a 1 second interval. Results for the first experiment are presented in figure 2.

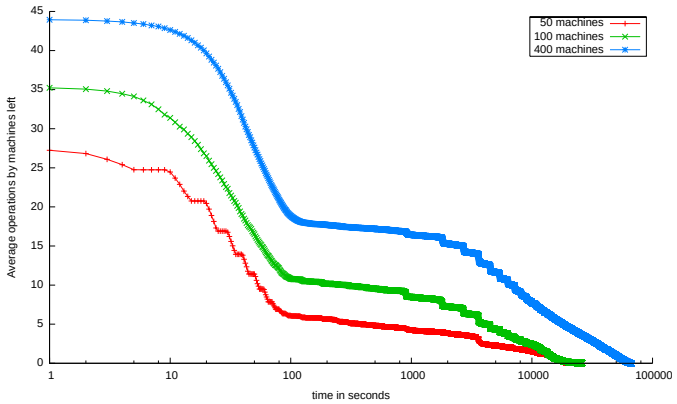

Figure 2: Average number of changes left by machine

It can be seen that even for small rings, composed of as few as 50 machines out of a possible 2^{128}, the time for the system to stabilize is large (over 5 hours). This time increases with the numbers of machines and can still be over 18h for a ring as small as 400 machines.

To reduce the duration of the experiments, we made use of the fact that a majority of changes in the routing tables are made in the first few seconds of initialization. It appears that after only 100s more than 50% of the changes have been made. Thus we do not wait for the whole system to be stabilized before injecting the first failure, but we wait for the whole system to have made enough changes in the routing tables and for it to be in a relatively steady state. Thus, the failure is injected 45min after the beginning of the experiment.

We call D-node the node we suppress from the ring, either by killing the process or destroying the machine. After suppressing the D-node we wait for 20 min for the nodes to update their routing tables. After this period we collect the routing tables and look for those which include the D-node. In those particular tables we search for the update that will make the D-node disappear from the routing table.

Each dot in figure 3 represents the update of the routing table of process y, at a time x, concerning the D-node. The "cross" dots represent the modifications before the failure is injected, thus the modifications due to the normal stabilization of FreePastry. The "plus" dots represent the modifications after the injection of the failure for the D-node in the case of process kill, and the last dots depict the case of virtual host destruction. The vertical line represents the date of the failure injection at the D-node (45 minutes after the beginning).

The set of routing tables that include the D-node consists of 578 nodes over several experiments. In this set many nodes delete the D-node of their routing table before it is suppressed. As it can be seen on the figure, all these nodes

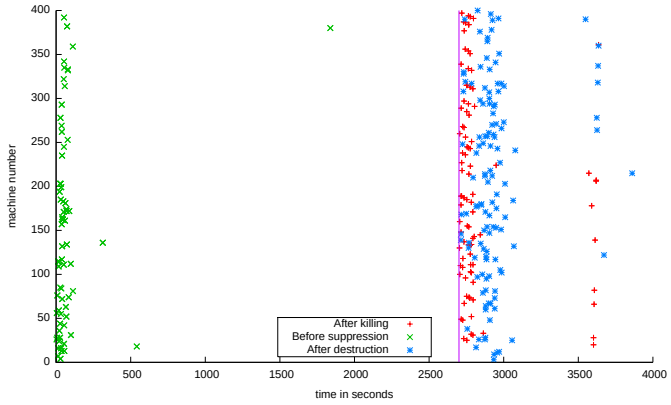

Figure 3: D-node deletion time

do it very early in the stabilization and therefore we can consider that every node that deletes the D-node from its table after the suppression time does it thanks to the failure detection component of Pastry.

Since the routing table maintenance is done lazily in Pastry [5], it is natural that not every node updates its routing table, since in the experiments no messages are exchanged.

When we only kill the pastry process to suppress the D-node after 45 minutes we can see on figure 3 that a lot of nodes react in a very short period of time to the suppression of the D-node. Comparing the points distributions for kill and destruction, we can see that nodes detect the failure in a shorter period of time in the case of kill than in the case of destruction. Since behaviors in the two cases are different we can consider that "destroying" a machine is more accurate since the stressed application must rely on its own failure detection mechanism, and the behavior of this application may be influenced by the asynchronism and the timings of the failure detection mechanism used. The figure also demonstrates that the active failure detection mechanism of FreePastry is effective and the distributed hash table is able to stabilize even with accurate failure injection.

5. CONCLUSION AND FUTURE WORK

In this paper, we presented an emulation platform for grids where both the machines and the network are virtualized at a low level. This allows an experimenter to test realistic failure injection into applications running on distributed architectures, such as grids. We evaluated the interest of our approach by running a classical fault-tolerant distributed application: Pastry.

We are in the process of developping a fault injection tool to work with the platform. The interest of this work is that using Xen virtual machines will allow to model strong adversaries through virtual machines with shared memory.

Acknowledgements..

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, an initiative from the French Ministry of Research through the ACI GRID incentive action, INRIA, CNRS, RENATER and other contributing partners.

6. REFERENCES

[1] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *SIGOPS Oper. Syst. Rev.*, 37(5):164–177, 2003.

[2] Erik Buchmann and Klemens Böhm. How to run experiments with large peer-to-peer data structures. *Parallel and Distributed Processing Symposium, International*, 1:27b, 2004.

[3] Franck Cappello, Eddy Caron, Michel Dayde, Frederic Desprez, Emmanuel Jeannot, Yvon Jegou, Stephane Lanteri, Julien Leduc, Nouredine Melab, Guillaume Mornet, Raymond Namyst, Pascale Primet, and Olivier Richard. Grid'5000: a large scale, reconfigurable, controlable and monitorable Grid platform. In *SC'05: Proc. The 6th IEEE/ACM International Workshop on Grid Computing Grid'2005*, pages 99–106, Seattle, USA, November 13-14 2005. IEEE/ACM.

[4] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, and Dan S. Wallach. Security for structured peer-to-peer overlay networks. In *5th Symposium on Operating Systems Design and Implementaion (OSDI'02)*, December 2002.

[5] Miguel Castro, Peter Druschel, Y. Charlie Hu, and Antony I. T. Rowstron. Topology-aware routing in structured peer-to-peer overlay networks. In André Schiper, Alexander A. Shvartsman, Hakim Weatherspoon, and Ben Y. Zhao, editors, *Future Directions in Distributed Computing*, volume 2584 of *Lecture Notes in Computer Science*, pages 103–107. Springer, 2003.

[6] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. Planetlab: an overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.*, 33(3):3–12, 2003.

[7] Andreas Haeberlen, Alan Mislove, Ansley Post, and Peter Druschel. Fallacies in evaluating decentralized systems. In *In Proceedings of IPTPS*, 2006.

[8] R. Housley and S. Hollenbeck. EtherIP: Tunneling Ethernet Frames in IP Datagrams. RFC 3378 (Informational), September 2002.

[9] Ralph Niederberger. DEISA: Motivations, strategies, technologies. In *Proc. of the Int. Supercomputer Conference (ISC'04)*, 2004.

[10] Lucas Nussbaum and Olivier Richard. Lightweight emulation to study peer-to-peer systems. *Concurr. Comput. : Pract. Exper.*, 20(6):735–749, 2008.

[11] Benjamin Quétier, Vincent Neri, and Franck Cappello. Scalability Comparison of Four Host Virtualization Tools. *Journal of Grid Computing*, 5:83–98, 2006.

[12] Luigi Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *SIGCOMM Comput. Commun. Rev.*, 27(1):31–41, 1997.

[13] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, November 2001.

[14] Quinn O. Snell, Armin R. Mikler, and John L. Gustafson. Netpipe: A network protocol independent performace evaluator. In *In Proceedings of the IASTED International Conference on Intelligent Information Management and Systems*, 1996.

[15] Amin Vahdat, Ken Yocum, Kevin Walsh, Priya Mahadevan, Dejan Kosti 'c, Jeff Chase, and David Becker. Scalability and accuracy in a large-scale network emulator. In *OSDI '02: Proceedings of the 5th symposium on Operating systems design and i mplementation*, pages 271–284, New York, NY, USA, 2002. ACM Press.

[16] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):255–270, 2002.