Testbeds in Computer Science

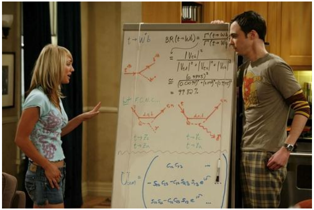
Lucas Nussbaum[1]

Reproducible Research Webinars, Episode IX

June 2017

[1] The Grid'5000 part is joint work with S. Delamare, F. Desprez, E. Jeanvoine, A. Lebre, L. Lefevre, D. Margery, P. Morillon, P. Neyron, C. Perez, O. Richard and many others

Validation in (Computer) Science



- Two classical approaches for validation:
  - Formal: equations, proofs, etc.
  - Experimental, on a scientific instrument
- Often a mix of both:
  - In Physics, Chemistry, Biology, etc.
  - In Computer Science

Distributed computing: a peculiar field in CS

- Performance and scalability are central to results
  - But depend greatly on the environment (hardware, network, software stack, etc.)
  - Many contributions are about *fighting* the environment
    - Making the most out of limited, complex and different resources (e.g. memory/storage hierarchy, asynchronous communications)
    - Handling performance imbalance, noise $\leadsto$ asynchronism, load balancing
    - Handling faults $\leadsto$ fault tolerance
    - Hiding complexity $\leadsto$ abstractions: middlewares, runtimes

Distributed computing: a peculiar field in CS

- Performance and scalability are central to results
  - But depend greatly on the environment (hardware, network, software stack, etc.)
  - Many contributions are about *fighting* the environment
    - Making the most out of limited, complex and different resources (e.g. memory/storage hierarchy, asynchronous communications)
    - Handling performance imbalance, noise $\leadsto$ asynchronism, load balancing
    - Handling faults $\leadsto$ fault tolerance
    - Hiding complexity $\leadsto$ abstractions: middlewares, runtimes
- Validation of most contributions require experiments
  - Formal validation often intractable or unsuitable
  - Even for more theoretical work $\leadsto$ simulation (SimGrid, CloudSim)

Distributed computing: a peculiar field in CS

- Performance and scalability are central to results
  - But depend greatly on the environment (hardware, network, software stack, etc.)
  - Many contributions are about *fighting* the environment
    - Making the most out of limited, complex and different resources (e.g. memory/storage hierarchy, asynchronous communications)
    - Handling performance imbalance, noise $\rightsquigarrow$ asynchronism, load balancing
    - Handling faults $\rightsquigarrow$ fault tolerance
    - Hiding complexity $\rightsquigarrow$ abstractions: middlewares, runtimes
- Validation of most contributions require experiments
  - Formal validation often intractable or unsuitable
  - Even for more theoretical work $\rightsquigarrow$ simulation (SimGrid, CloudSim)
- But experimenting is difficult and time-consuming… but often neglected
  - *Everybody is doing it, not so many people are talking about it*

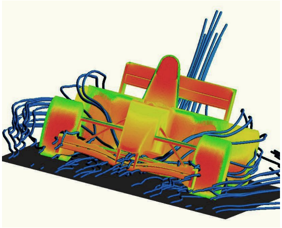
This talk

1. Panorama: experimental methodologies, tools, testbeds

2. Grid'5000: a large-scale testbed for distributed computing

Experimental methodologies

Simulation



1. **Model** application
2. **Model** environment
3. **Compute** interactions

Real-scale experiments

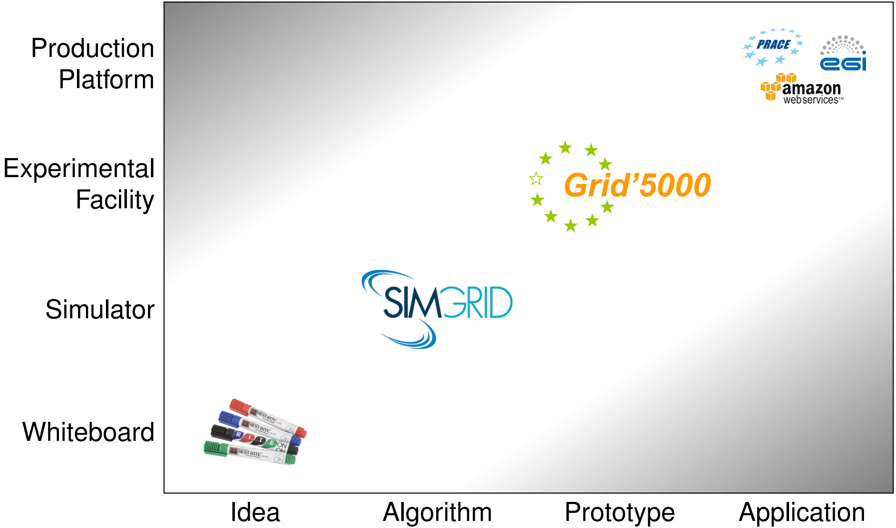
Execute the **real** application on **real** machines

Complementary solutions:

☺ Work on algorithms
☺ More scalable, easier

☺ Work on applications
☺ Perceived as more realistic

From ideas to applications



Example testbed: PlanetLab (2002 → ~2012)[2]

PLANETLAB

- 700-1000 nodes (generally two per physical location)
- Users get *slices*: sets of virtual machines
- Heavily used to study network services, P2P, network connectivity
- Limitations:
  - Shared nodes (varying & low computation power)
  - "Real" Internet:
    - Unstable experimental conditions
    - Nodes mostly connected to GREN ⇝ not really representative

[2] Brent Chun et al. “Planetlab: an overlay testbed for broad-coverage services”. In: *ACM SIGCOMM Computer Communication Review* 33.3 (2003), pages 3–12.

Experimental methodologies (2)

A more complete picture[3]:

		Environment	
		Real	*Model*
Application	*Real*	**In-situ** (Grid'5000, DAS3, PlanetLab, GINI, ...)	**Emulation** (Microgrid, Wrekavock, V-Grid, Dummynet, TC, ...)
	Model	**Benchmarking** (SPEC, Linpack, NAS, IOzone, ...)	**Simulation** (SimGRID, GRIDSim, NS2, PeerSim, P2PSim, DiskSim, ...)

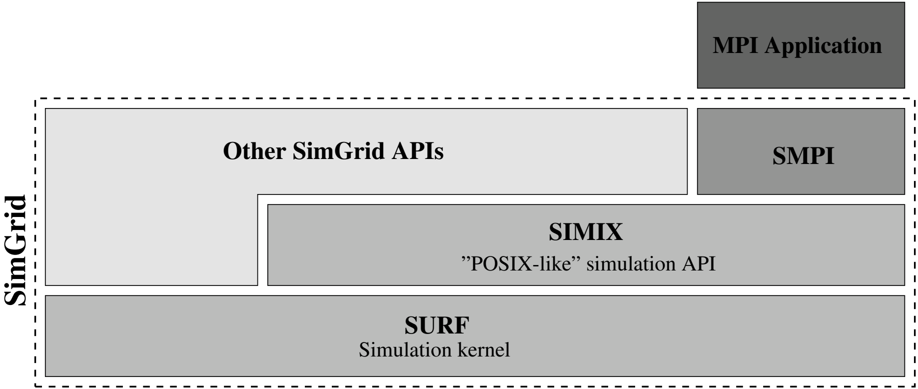
Two approaches for emulation:

- Start from a simulator, add API to execute unmodified applications
- Start from a real testbed, alter (degrade performance, virtualize)

[3] Jens Gustedt, Emmanuel Jeannot, and Martin Quinson. "Experimental Methodologies for Large-Scale Systems: a Survey". In: *Parallel Processing Letters* 19.3 (2009), pages 399–418.

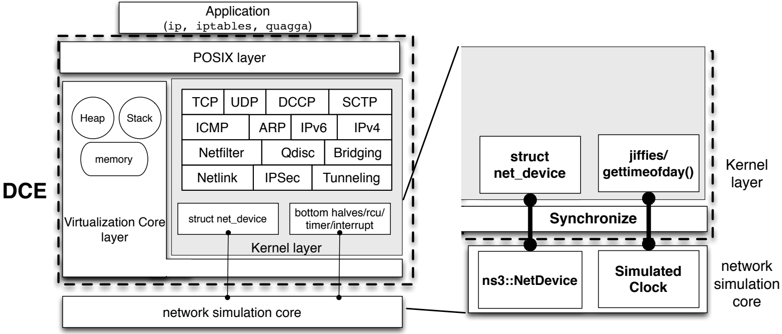
Emulator on top of a simulator: SMPI[4]

- SimGrid-backed MPI implementation
- Run MPI application on simulated cluster with `smpicc` ; `smpirun`



[4] Pierre-Nicolas Clauss et al. "Single node on-line simulation of MPI applications with SMPI". In: *International Parallel & Distributed Processing Symposium*. 2011, pages 664–675.

Emulator on top of the NS3 simulator: DCE[5]



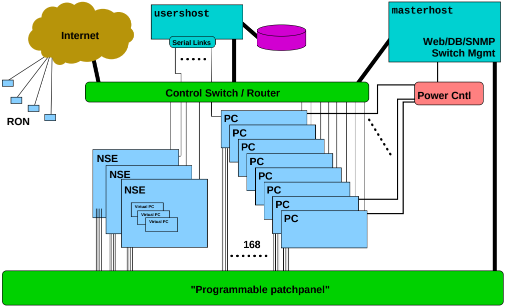
- Virtualization layer to manage resources for each instance (inside a single Linux process)
- POSIX layer to emulate relevant *libc* functions (404 supported) to execute unmodified Linux applications

[5] Hajime Tazaki et al. "Direct code execution: Revisiting library os architecture for reproducible network experiments". In: *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. 2013, pages 217–228.

2nd approach: emulator on top of a real system

- Take a real system
- Degrade it to make it match experimental conditions

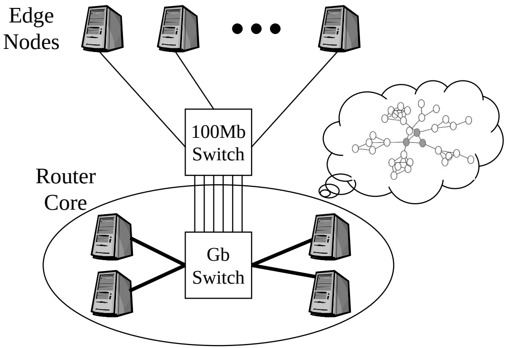
Network emulation: Emulab[6]



- Use a cluster of nodes with many network interfaces
- Configure the network on the fly to create custom topologies
  - With link impairement (latency, bandwidth limitation)
- Emulab: a testbed at Univ. Utah, and a software stack
  - Deployed on dozens of testbed world-wide (inc. CloudLab)
    In Europe: IMEC's Virtual Wall (Ghent, Belgium)

[6] Brian White et al. "An integrated experimental environment for distributed systems and networks". In: *ACM SIGOPS Operating Systems Review* 36.SI (2002), pages 255–270.

Network emulation: Modelnet[7]

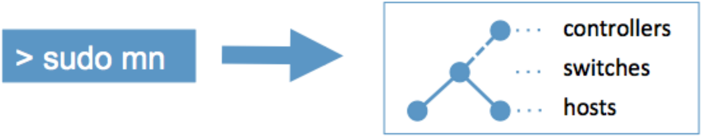


- Similar principle: let a cluster of nodes handle the network emulation

[7] Amin Vahdat et al. “Scalability and accuracy in a large-scale network emulator”. In: *ACM SIGOPS Operating Systems Review* 36.SI (2002), pages 271–284.

Network emulation: Mininet[8]

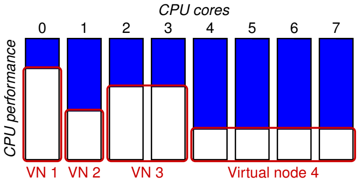
- Everything on a single Linux system
- Using containers technology (*netns*), Linux TC/netem, OpenVSwitch
- Hugely popular in the networking community due to ease of use



[8] Bob Lantz, Brandon Heller, and Nick McKeown. "A network in a laptop: rapid prototyping for software-defined networks". In: *9th ACM SIGCOMM Workshop on Hot Topics in Networks*. 2010.

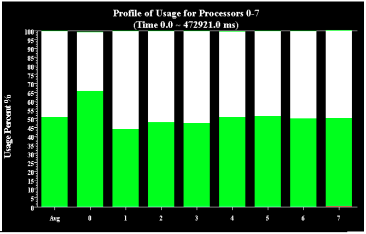
CPU performance emulation: Distem[9]

- Reduce available CPU time using various techniques
  (CPU burner, scheduler tuning, CPU frequency scaling)

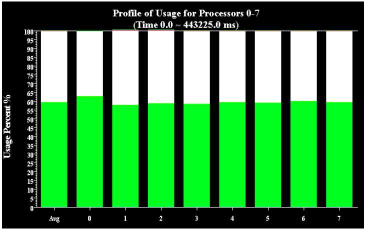


- Example: testing Charm++ load balancing

No load balancing (time: 473s)



RefineLB (time: 443s)



[9] Luc Sarzyniec, Tomasz Buchert, Emmanuel Jeanvoine, and Lucas Nussbaum. "Design and evaluation of a virtual experimental environment for distributed systems". In: *PDP*. 2013.

Time dilation: DieCast[10]

- Problem: when degrading performance, one can only get slower-than-real performance
- Idea: slow down the time by a *time dilation factor*
- Result: hardware looks faster

TDF	Real Configuration	Perceived Configuration
1	100 Mbps, 80 ms	100 Mbps, 80 ms
10	100 Mbps, 80 ms	1000 Mbps, 8 ms
10	10 Mbps, 800 ms	100 Mbps, 80 ms
1	B Mbps, L ms	B Mbps, L ms
t	B/t Mbps, $L \times t$ ms	B Mbps, L ms

[10] Diwaker Gupta et al. "DieCast: Testing distributed systems with an accurate scale model". In: *ACM Transactions on Computer Systems (TOCS)* 29.2 (2011), page 4.

Testbeds

Difficult to survey:

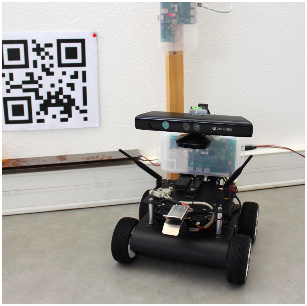
- Moving targets (papers often outdated, need to look at tutorials or papers using the testbed)
- Both scientific objects and scientific instruments, with their own life

Typical questions:

- What kind of resources are provided? (target fields)
- How much can the experimenter control? (what can be changed?)
- What kind of guarantees are provided about the environment?
- What additional services are provided (e.g. monitoring)?
- What is the interface (API) to use the testbed?
- What is the current status ? (churn due to project-based funding)

Internet of Things: FIT IoT-Lab[11]

- 2769 wireless sensors (from WSN430 to Cortex A8)
- 7 sites (Grenoble, Lille, Strasbourg, Saclay, Rennes, IMT Paris, Lyon)
- Also mobile robots
- Typical experiment: IoT communication protocols

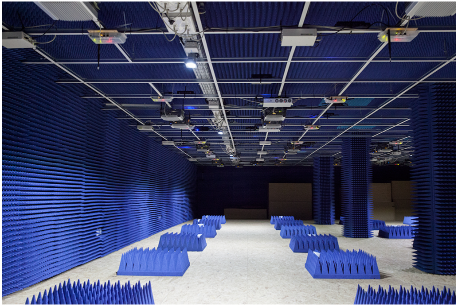


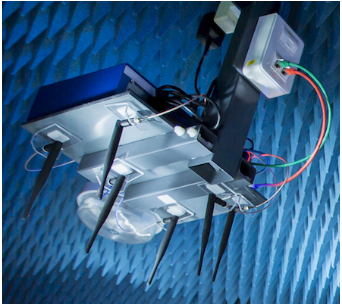
`https://www.iot-lab.info/`

[11] Cedric Adjih et al. "FIT IoT-LAB: A large scale open experimental IoT testbed". In: *IEEE 2nd World Forum on Internet of Things (WF-IoT)*. 2015.

Wireless (WiFi, 4G/LTE, SDR): CorteXlab[12], R2lab

- Sets of customizable wireless nodes in an anechoic chamber
- For experiments on the physical layer





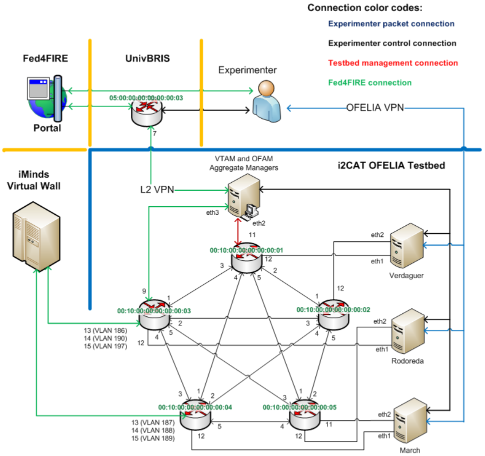
`http://www.cortexlab.fr`
`https://r2lab.inria.fr`

[12] Albdelbassat Massouri et al. "CorteXlab: An Open FPGA-based Facility for Testing SDR & Cognitive Radio Networks in a Reproducible Environment". In: *INFOCOM'2014 Demo/Poster Session*. 2014.

Software Defined Networking: OFELIA[13]

- Set of sites (*islands*), each site hosts OpenFlow-enabled switches
- Users control their OpenFlow controller, and VM to act as sources/sinks

OFELIA Facility and Islands



[13] Marc Suñé et al. "Design and implementation of the OFELIA FP7 facility: The European OpenFlow testbed". In: *Computer Networks* 61 (2014), pages 132–150.

Clouds, data centers

- Grid’5000, Emulab/Cloudlab, Chameleon
- Discussed in the second part of this talk

Federations of testbeds

- **Identity-level federation**
  - Enable users to use several testbeds with same credentials
- **API-level federation**
  - Provide the same interface on/for several testbeds
- **Data-plane federation**
  - Combine resources from several testbeds during an experiment
  - Two main use cases:
    - Different testbeds (e.g. Cloud/Edge scenarios, with experiment control at both ends)
    - Similar testbeds $\leadsto$ more resources, more distributed

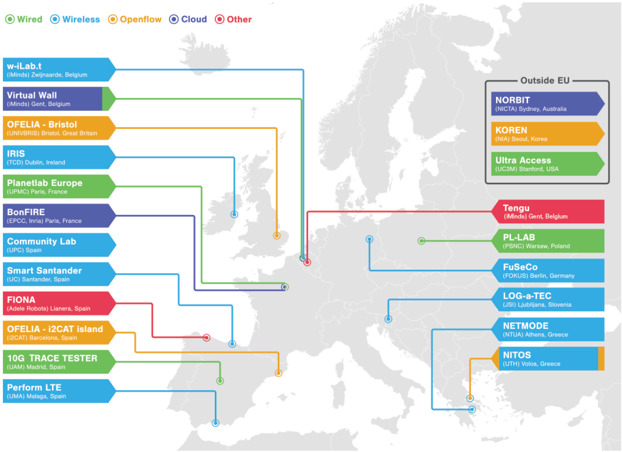
GENI[14]

- The flagship project of testbed federation
- A large-scale distributed testbed, or a tightly integrated federation of ***aggregates***, providing either compute resources (*racks*) or networking
  - InstaGENI racks (32 currently):
    - Descendant from the Emulab software stack
    - Providing VMs (Xen) or raw PCs
    - HP hardware
  - ExoGENI racks (12 currently):
    - VMs using OpenStack, or Xen, or OpenVZ
    - Some racks with bare-metal nodes (xCAT)
    - IBM hardware
  - AL2S, MAX: providing network interconnection between racks
- Also the main developer of the GENI API, used by other federations

[14] Rick McGeer, Mark Berman, Chip Elliott, and Robert Ricci. *The GENI Book*. 1st. Springer Publishing Company, Incorporated, 2016. ISBN: 331933767X, 9783319337678.

Fed4FIRE

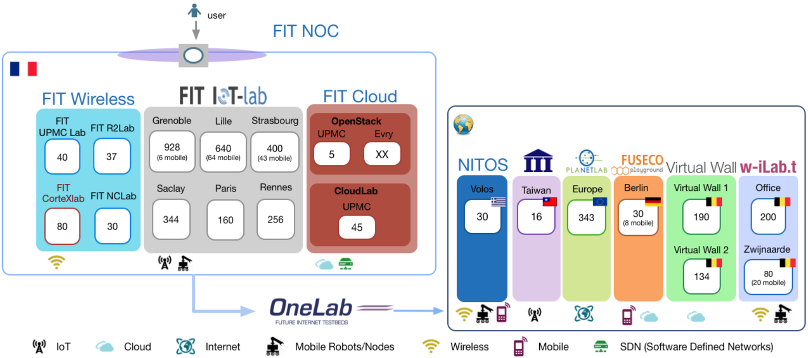
- European federation of about 20 testbeds
- Diverse: wired networking, wireless/5G, IoT, OpenFlow, Cloud



https://www.fed4fire.eu/

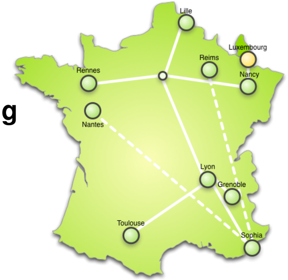
FIT federation

- French federation of testbeds, funded by Equipex
- Gathers:
  - An IoT testbed: FIT IoT-Lab
  - Wireless testbeds: FIT CorteXLab, FIT UPMC Lab, FIT R2Lab, FIT NC Lab
  - Cloud testbeds: two OpenStack instances, one Emulab instance
  - A unified portal (OneLab)

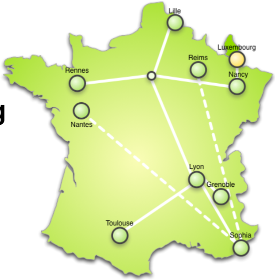


The Grid'5000 testbed

- **A large-scale testbed for distributed computing**
  - 8 sites, 30 clusters, 840 nodes, 8490 cores
  - Dedicated 10-Gbps backbone network
  - 550 users and 100 publications per year



The Grid'5000 testbed



- **A large-scale testbed for distributed computing**
  - 8 sites, 30 clusters, 840 nodes, 8490 cores
  - Dedicated 10-Gbps backbone network
  - 550 users and 100 publications per year
- A meta-grid, meta-cloud, meta-cluster, meta-data-center:
  - Used by CS researchers in HPC / Clouds / Big Data / Networking
  - To experiment in a fully controllable and observable environment
  - Similar problem space as Chameleon and Cloudlab (US)
  - Design goals:
    - Support high-quality, reproducible experiments
    - On a large-scale, shared infrastructure

Landscape – cloud & experimentation

- Public cloud infrastructures (AWS, Azure, Google, etc.)
  - ☹ No information/guarantees on placement, multi-tenancy, real performance
- Private clouds: Shared observable infrastructures
  - ☺ Monitoring & measurement
  - ☹ No control over infrastructure settings
  - $\rightsquigarrow$ Ability to understand experiment results
- On-demand clouds – dedicated observable infrastructures (BonFIRE)
  - ☺ Limited ability to alter infrastructure
- Bare-metal as a service, fully reconfigurable infrastructure (Grid'5000)
  - ☺ Control/alter all layers, including virtualization technology, operating system, networking

Outline

1. Discovering resources from their description
2. Reconfiguring the testbed to meet experimental needs
3. Monitoring experiments, extracting and analyzing data
4. Data management
5. Improving control and description of experiments

Discovering resources from their description

- Describing resources ⇝ understand results
  - Covering nodes, network equipment, topology
  - Machine-parsable format (JSON) ⇝ scripts
  - Archived (*State of testbed 6 months ago?*)


```
"processor": {
  "cache_l2": 8388608,
  "cache_l1": null,
  "model": "Intel Xeon",
  "instruction_set": "",
  "other_description": "",
  "version": "X3440",
  "vendor": "Intel",
  "cache_l1i": null,
  "cache_l1d": null,
  "clock_speed": 2530000000.0
},
"uid": "graphene-1",
"type": "node",
"architecture": {
  "platform_type": "x86_64",
  "smt_size": 4,
  "smp_size": 1
},
"main_memory": {
  "ram_size": 17179869184,
  "virtual_size": null
},
"storage_devices": [
  {
    "model": "Hitachi HDS72103",
    "size": 298023223876.953,
    "driver": "ahci",
    "interface": "SATA II",
    "rev": "JPFO",
    "device": "sda"
  }
],
```

Discovering resources from their description

- Describing resources ⇝ understand results
  - Covering nodes, network equipment, topology
  - Machine-parsable format (JSON) ⇝ scripts
  - Archived (*State of testbed 6 months ago?*)
- Verifying the description
  - Avoid inaccuracies/errors ⇝ wrong results
  - Could happen frequently: maintenance, broken hardware (e.g. RAM)
  - Our solution: g5k-checks
    - Runs at node boot (or manually by users)
    - Acquires info using OHAI, ethtool, etc.
    - Compares with Reference API


```
"processor": {
  "cache_l2": 8388608,
  "cache_l1": null,
  "model": "Intel Xeon",
  "instruction_set": "",
  "other_description": "",
  "version": "X3440",
  "vendor": "Intel",
  "cache_l1i": null,
  "cache_l1d": null,
  "clock_speed": 2530000000.0
},
"uid": "graphene-1",
"type": "node",
"architecture": {
  "platform_type": "x86_64",
  "smt_size": 4,
  "smp_size": 1
},
"main_memory": {
  "ram_size": 17179869184,
  "virtual_size": null
},
"storage_devices": [
  {
    "model": "Hitachi HDS72103",
    "size": 298023223876.953,
    "driver": "ahci",
    "interface": "SATA II",
    "rev": "JPFO",
    "device": "sda"
  }
],
```

Discovering resources from their description

- Describing resources $\rightsquigarrow$ understand results
  - Covering nodes, network equipment, topology
  - Machine-parsable format (JSON) $\rightsquigarrow$ scripts
  - Archived (*State of testbed 6 months ago?*)
- Verifying the description
  - Avoid inaccuracies/errors $\rightsquigarrow$ wrong results
  - Could happen frequently: maintenance, broken hardware (e.g. RAM)
  - Our solution: g5k-checks
    - Runs at node boot (or manually by users)
    - Acquires info using OHAI, ethtool, etc.
    - Compares with Reference API
- Selecting resources
  - OAR database filled from Reference API

```
oarsub -p "wattmeter='YES' and gpu='YES'"
oarsub -l "cluster='a'/nodes=1+cluster='b' and
      eth10g='Y'/nodes=2,walltime=2"
```


```
"processor": {
  "cache_l2": 8388608,
  "cache_l1": null,
  "model": "Intel Xeon",
  "instruction_set": "",
  "other_description": "",
  "version": "X3440",
  "vendor": "Intel",
  "cache_l1i": null,
  "cache_l1d": null,
  "clock_speed": 2530000000.0
},
"uid": "graphene-1",
"type": "node",
"architecture": {
  "platform_type": "x86_64",
  "smt_size": 4,
  "smp_size": 1
},
"main_memory": {
  "ram_size": 17179869184,
  "virtual_size": null
},
"storage_devices": [
  {
    "model": "Hitachi HDS72103",
    "size": 298023223876.953,
    "driver": "ahci",
    "interface": "SATA II",
    "rev": "JPF0",
    "device": "sda"
  }
],
```

Reconfiguring the testbed

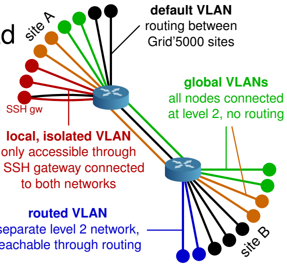
- Typical needs:
  - Install specific software
  - Modify the kernel
  - Run custom distributed middlewares (Cloud, HPC, Grid)
  - Keep a stable (over time) software environment

Reconfiguring the testbed

- Typical needs:
  - Install specific software
  - Modify the kernel
  - Run custom distributed middlewares (Cloud, HPC, Grid)
  - Keep a stable (over time) software environment
- Likely answer on any production facility: **you can't**
- Or:
  - Install in $HOME, modules $\rightsquigarrow$ no root access, handle custom paths
  - Use virtual machines $\rightsquigarrow$ experimental bias (performance), limitations
  - Containers: kernel is shared $\rightsquigarrow$ various limitations

Reconfiguring the testbed

- Operating System reconfiguration with Kadeploy:
  - Provides a *Hardware-as-a-Service* cloud infrastructure
  - Enable users to deploy their own software stack & get *root* access
  - **Scalable, efficient, reliable and flexible**:
    200 nodes deployed in ~5 minutes
- Customize networking environment with KaVLAN
  - Protect the testbed from experiments (Grid/Cloud middlewares)
  - Avoid network pollution
  - Create custom topologies
  - By reconfiguring VLANS $\rightsquigarrow$ almost no overhead



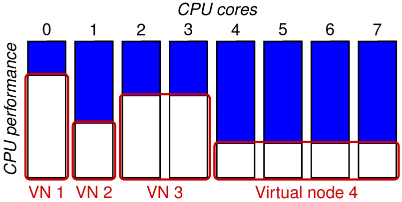
Creating and sharing Kadeploy images

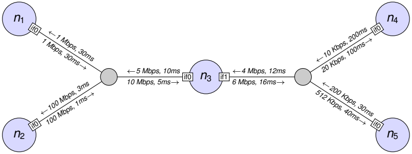
- When doing manual customization:
  - Easy to forget some changes
  - Difficult to describe
  - The full image must be provided
  - Cannot really serve as a basis for future experiments (similar to binary vs source code)
- Kameleon: Reproducible generation of software appliances
  - Using *recipes* (high-level description)
  - Persistent cache to allow re-generation without external resources (Linux distribution mirror) $\rightsquigarrow$ self-contained archive
  - Supports Kadeploy images, LXC, Docker, VirtualBox, qemu, etc.

http://kameleon.imag.fr/

Changing experimental conditions

- Reconfigure experimental conditions with Distem
  - Introduce heterogeneity in an homogeneous cluster
  - Emulate complex network topologies



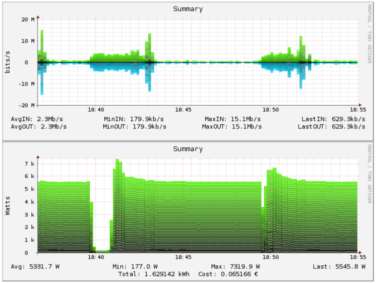


http://distem.gforge.inria.fr/

Monitoring experiments

Goal: enable users to understand what happens during their experiment

- System-level probes (usage of CPU, memory, disk, with Ganglia)
- Infrastructure-level probes
  - Network, power consumption
  - Captured at high frequency ($\approx$1 Hz)
  - Live visualization
  - REST API
  - Long-term storage



Data management

- Already available: file-based and block-based storage
  - Storage5k
  - Managed Ceph clusters in Rennes and Nantes
  - OSIRIM: large storage space made available by the OSIRIM project in Toulouse
- Currently in beta: reservation of disks on nodes, to store large datasets between nodes reservations
- **Missing: long-term archival of experiment data**
  - Probably not a good idea to solve this on our own
    $\rightsquigarrow$ Data repository sponsored by Inria, CNRS, or another institution?

Improving control and description of experiments

- Legacy way of performing experiments: shell commands
  - ☹ time-consuming
  - ☹ error-prone
  - ☹ details tend to be forgotten over time

- Promising solution: automation of experiments
  $\rightsquigarrow$ Executable description of experiments

- Similar problem-space as *configuration mgmt*, *infrastructure as code*
  - ♦ But not just the initial setup

- Support from the testbed: Grid'5000 RESTful API
  (Resource selection, reservation, deployment, monitoring)

Tools for automation of experiments

Several projects around Grid'5000 (but not specific to Grid'5000):

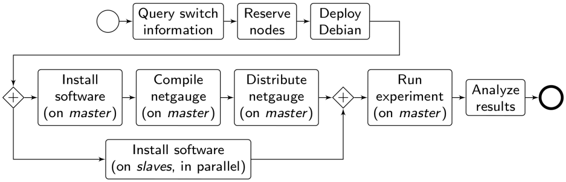
- g5k-campaign (Grid'5000 tech team)
- Expo (Cristian Ruiz)
- Execo (Mathieu Imbert)
- XPFlow (Tomasz Buchert)

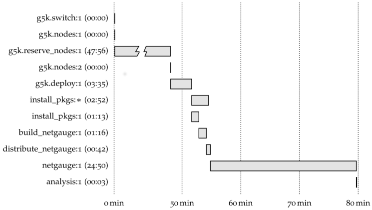
Features:

- Facilitate scripting of experiments in high-level languages (Ruby, Python)
- Provide useful and efficient abstractions :[15]
  - Testbed management
  - Local & remote execution of commands
  - Data management
- *Engines* or *workflows* for more complex processes

[15] Tomasz Buchert, Cristian Ruiz, Lucas Nussbaum, and Olivier Richard. "A survey of general-purpose experiment management tools for distributed systems". In: *Future Generation Computer Systems* 45 (2015), pages 1–12.

XPFlow[16]






```
engine.process :exp do |site, switch|
    s = run g5k.switch, site, switch
    ns = run g5k.nodes, s
    r = run g5k.reserve_nodes,
        :nodes => ns, :time => '2h',
        :site => site, :type => :deploy
    master = (first_of ns)
    rest = (tail_of ns)
    run g5k.deploy,
        r, :env => 'squeeze-x64-nfs'
    checkpoint :deployed
    parallel :retry => true do
        forall rest do |slave|
            run :install_pkgs, slave
        end
        sequence do
            run :install_pkgs, master
            run :build_netgauge, master
            run :dist_netgauge,
                master, rest
        end
    end
    checkpoint :prepared
    output = run :netgauge, master, ns
    checkpoint :finished
    run :analysis, output, switch
end
```

Experiment description and execution as a Business Process Workflow

Supports parallel execution of activities, error handling, snapshotting, built-in logging and provenance collection, etc.

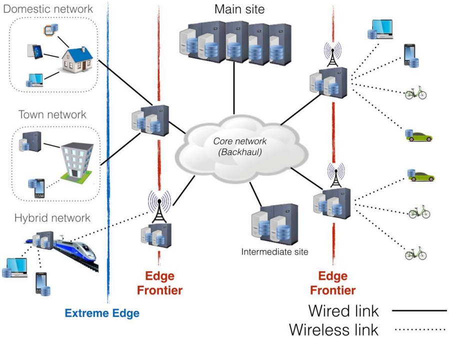
[16] Tomasz Buchert. "Managing large-scale, distributed systems research experiments with control-flows". PhD Thesis. Université de Lorraine, Jan. 2016.

What's new?

- New clusters in Lille and Lyon
- Minor improvement (but important for usability): OAR job extensions
- Large storage space available in Toulouse (OSIRIM project)
- Reserve disks on nodes to store datasets between nodes reservations
- Preparation for the Debian 9 *Stretch* release (*stretch-x64-min* available)

What's next?

- New clusters in 2017: Nancy (deep learning), Nantes (energy), Lille and Grenoble (HPC)
- Improved Docker support (soon)
- Federation (Fed4FIRE+ EU project, 2017-2022)
- SILECS project:
  - Grid'5000 and FIT merge
  - A new infrastructure for large-scale experimental computer science



Conclusions

- Grid'5000: a testbed for high-quality, reproducible research on HPC, Clouds, Big Data and Networking
- With a unique combination of features
  - Description and verification of testbed
  - Reconfiguration (hardware, network)
  - Monitoring
  - Support for automation of experiments
- Try it yourself!
  - Free account through the Open Access program
    `http://www.grid5000.fr/open-access`
  - Tutorials available on the website (and later this month at COMPAS)

`https://www.grid5000.fr`

Bibliography

- Resources management: Resources Description, Selection, Reservation and Verification on a Large-scale Testbed. http://hal.inria.fr/hal-00965708
- Kadeploy: Kadeploy3: Efficient and Scalable Operating System Provisioning for Clusters. http://hal.inria.fr/hal-00909111
- KaVLAN, Virtualization, Clouds deployment:
  - Adding Virtualization Capabilities to the Grid'5000 testbed. http://hal.inria.fr/hal-00946971
  - Enabling Large-Scale Testing of IaaS Cloud Platforms on the Grid'5000 Testbed. http://hal.inria.fr/hal-00907888
- Kameleon: Reproducible Software Appliances for Experimentation. https://hal.inria.fr/hal-01064825
- Distem: Design and Evaluation of a Virtual Experimental Environment for Distributed Systems. https://hal.inria.fr/hal-00724308
- XP management tools:
  - A survey of general-purpose experiment management tools for distributed systems. https://hal.inria.fr/hal-01087519
  - XPFlow: A workflow-inspired, modular and robust approach to experiments in distributed systems. https://hal.inria.fr/hal-00909347
  - Using the EXECO toolbox to perform automatic and reproducible cloud experiments. https://hal.inria.fr/hal-00861886
  - Expo: Managing Large Scale Experiments in Distributed Testbeds. https://hal.inria.fr/hal-00953123
- Kwapi: A Unified Monitoring Framework for Energy Consumption and Network Traffic. https://hal.inria.fr/hal-01167915