Testing Testbeds Towards Reproducibility

Lucas Nussbaum
lucas.nussbaum@loria.fr

GEFI workshop 2017

Context: the Grid'5000 testbed

- **A large-scale distributed testbed for distributed computing**
  - 8 sites, 32 clusters, 894 nodes, 8490 cores
  - Dedicated 10-Gbps backbone network
  - 550 users and 100 publications per year

Context: the Grid'5000 testbed

- **A large-scale distributed testbed for distributed computing**
  - 8 sites, 32 clusters, 894 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
  - **Design goals:**
    - ★ **Support high-quality, reproducible experiments**
    - ★ **On a large-scale, shared infrastructure**

This talk: the Grid'5000 testing framework

Goals:

- Systematically test the Grid'5000 infrastructure and its services
- Increase the reliability and the trustworthiness of the testbed
- Uncover problems that would harm the repeatability and the reproducibility of experiments

Outline:

- Motivations for this work
- Our solution
- Results
- Conclusions

Grid'5000: overview

- Fairly used testbed
- Many services that support good-quality experiments
- Still, sometimes (rarely), scary bugs were found
  - Showing that some serious problems were not detected
- Problem: very few bugs are reported
  - Reporting bugs or asking technical questions is a difficult process[1][2]
    - Typical users of testbeds (students, post-docs) rarely have that skill
    - Or lack the confidence to report bugs

[1] Simon Tatham. "How to Report Bugs Effectively". 1999. URL: http://www.chiark.greenend.org.uk/~sgtatham/bugs.html.

[2] Eric Steven Raymond and Rick Moen. "How To Ask Questions The Smart Way". URL: http://www.catb.org/esr/faqs/smart-questions.html.

But many bugs should be reported

Several factors for many different and interesting issues:

- Scale: 8 sites, 32 clusters, 894 nodes
  - Not really a problem on the software side (config mgmt tools)
  - Hardware of different age, from different vendors
  - Hardware requiring some manual configuration
  - Hardware with silent and subtle failure patterns[3]
- Software stack
  - Some core services – well tested
  - But also experimental ones
    - Testbeds are always trying to innovate
    - But adoption generally slow

[3] https://youtu.be/tDacjrSCeq4?t=47s

Bugs can have dramatic consequences

- Most experiments focus on measuring performance
  - So subtle performance bugs can have a huge impact
  - 5% decrease in performance
    $\rightsquigarrow$ wrong results
    $\rightsquigarrow$ wrong conclusions
    $\rightsquigarrow$ retracted paper?
- Example bugs (all real):
  - Different CPU settings (power mgmt, hyperthreading, turbo boost)
  - Different disk firmware version, disk cache settings
  - Cabling issue $\rightsquigarrow$ wrong measurements by testbed monitoring service
- Problems on the software side
  $\rightsquigarrow$ unreliable services
  $\rightsquigarrow$ harder to automate experiments

Our testbed testing framework

- Based on Jenkins
- With custom developments
  - For job scheduling
  - For analyzing and summarizing results

Jenkins automation server

- De facto standard tool for automating processes (CI, CD)
  - `cron` on steroids
  - Extensible through plugins
    - Matrix Project: jobs as matrices of several options
      test_environments: 14 images X 32 clusters = 448 configurations
    - Matrix Reloaded: retry subset of configurations in Matrix jobs
- However, Jenkins alone was not sufficient for our needs

Job scheduling

- Basic scheduling available in Jenkins (time-based)
  - Not sufficient for our needs
- Different kinds of tests:
  - Software-centric: one node per cluster
  - Hardware-centric: all nodes of a given cluster
- Resources are heavily used
  - Waiting for all nodes of a given cluster to be available can take weeks
- One cannot just submit a job and wait because:
  - It would use a Jenkins *worker* (slot)
  - It would compete with user requests

Job scheduling (2)

- Implemented in an external tool that triggers Jenkins builds
- Queries the job status and the testbed status, and decides to submit a job based on:
  - Resources availability
  - Retry policy (exponential backoff)
  - Additional policies (peak hours, avoid several jobs on same site)
- If the Jenkins build creates a testbed job, but that testbed job fails to be scheduled immediately, it is cancelled and the build is marked as *unstable* in Jenkins

Analyzing and summarizing results

- Requirements:
  - Per test status, or all sites/clusters $\rightsquigarrow$ OK
  - Per site or per cluster status, for all tests
  - Historical perspective
- Solution: external status page that uses Jenkins' REST API

Analyzing and summarizing results

Site	Average	cmdline	console	disk	environments	kavlan	mpigraph	multideploy	multireboot	oarproperties	oarstate	paralleldeploy	refapi	sidapi	stdenv
grenoble	77%	100%	33%	33%	87%	33%	100%	66%	100%	100%	100%	0%	0%	100%	66%
lille	85%	100%	60%	40%	98%	80%	100%	100%	80%	60%	0%	100%	0%	100%	80%
luxembourg	95%	100%	100%	100%	100%	50%	100%	100%	100%	100%	100%	0%	100%	100%	100%
lyon	78%	100%	25%	50%	100%	50%	75%	100%	25%	0%	0%	100%	0%	100%	100%
nancy	92%	100%	100%	55%	99%	66%	95%	88%	33%	100%	100%	100%	100%	100%	88%
nantes	88%	100%	100%	0%	100%	0%	100%	100%	0%	100%	100%	100%	100%	100%	100%
rennes	89%	100%	60%	40%	98%	60%	100%	100%	60%	100%	100%	100%	60%	100%	100%
sophia	80%	100%	75%	0%	100%	25%	100%	75%	50%	100%	0%	100%	0%	100%	25%
Average	86%	100%	69%	42%	98%	54%	96%	90%	54%	81%	62%	75%	45%	100%	81%

Showing 1 to 9 of 9 entries

hide bugs with comments | reset

Search:

Job	Configuration	Status	Last successful	Last failed	Streak	Last attempt	Next	Comment (from pad)
test_disk	site_cluster=nancy-graphene	Fail		2017-01-18 07:50:45	7	2017-01-24 15:18:39	2017-01-24 16:18:39	NORETRY graphene-[45,48] ont des disques différents
test_disk	site_cluster=grenoble-edel	Fail		2017-01-19 19:30:38	11	2017-01-24 15:18:39	2017-01-24 16:18:39	NORETRY Bug 7696 Les disques du cluster Edel ne sont pas homogènes
test_disk	site_cluster=nancy-griffon	Fail		2017-01-25 15:00:56	11	2017-01-25 15:00:56	No retry	NORETRY Bug 7675 griffon : disks are not homogeneous
test_console	site_cluster=rennes-paravance	Fail		2017-01-27 07:30:57	7	2017-01-27 07:30:57	2017-02-03 07:30:57	Bug 7770 - kaconsole failed on paravance-56
test_disk	site_cluster=rennes-paravance	Fail		2017-01-20 07:00:41	10	2017-01-27 07:31:10	2017-01-27 07:00:41	Bug 7737 test-disk on paravance
test_multireboot	environment=jessie-x64-min,site_cluster=sophia-uvb	Fail		2017-01-02 15:10:40	1	2017-01-02 15:10:40	2017-01-09 15:10:40	Bug 7686 - uvb - some nodes fail on reboot
test_kavlan	site_cluster=grenoble-genepi	Fail	2016-12-02 02:10:45	2017-01-24 20:30:57	12	2017-01-24 20:30:57	2017-01-31 20:30:57	Bug 7685 - kavlan fail to put node in VLAN 100: Configuration session timed out!
test_refapi	site_cluster=rennes-parapluie	Fail		2017-01-27 12:01:15	34	2017-01-27 12:01:15	2017-02-03 12:01:15	Bug 7585 Homogénéité des clusters de Rennes

Test scripts

- Goals: exhibit issues, but also provide sufficient information to testbed operators to understand and fix the issue
- Coverage (total of 751 test configurations):
  - Homogeneity and correctness of testbed description (*refapi*, *oarproperties*, *dellbios*)
  - Testbed status (*oarstate*)
  - Basic functionality of command-line tools, REST API (*cmdline*, *sidapi*)
  - Provided system images (*environments*, *stdenv*)
  - Reliability of key services (*paralleldeploy*, *multireboot*, *multideploy*)
  - Other important services (*console*, *kavlan*, *kwapi*)
  - Specific hardware: Infiniband, hard disk drives (*mpigraph*, *disk*)

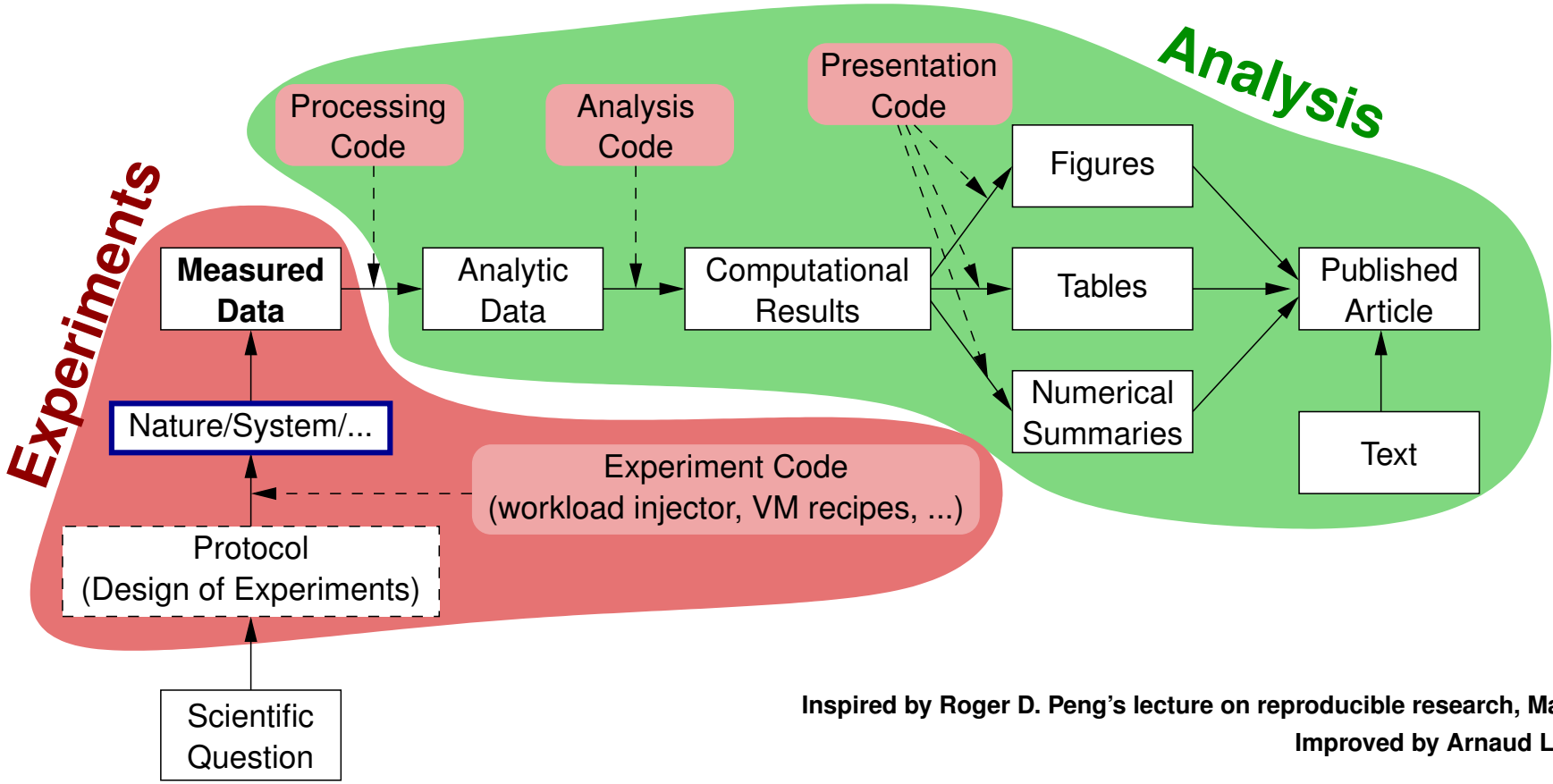
Results

- From October 2016 to April 2017: 118 bugs found
  - Disk drives configuration (R/W caching), CPU settings (C-states)
  - Different disk performance due to different disk firmware versions
  - Cabling issues
  - Various weak spots in the infrastructure, and configuration problems
  - A cluster was decommissioned after tests exhibited random reboots
  - Other random problems:
    - A race condition in the Linux kernel caused boot delays
    - A bug in the Infiniband stack caused random failures to start

Wrapping-up

- Testbed testing framework:
  - Systematically test the Grid'5000 infrastructure and its services
  - Increase the reliability and the trustworthiness of the testbed
  - Uncover problems that would harm the repeatability and the reproducibility of experiments
- Outcomes:
  - Many problems identified and fixed
  - Testbed reliability improving (85% of tests successful in February $\rightsquigarrow$ 93% today, despite the addition of new tests)
  - Impact on the way the testbed operators work $\rightsquigarrow$ *Test-driven operations*, more confidence that what should work actually works
  - Tests still being added
    - ★ Adding real user experiments as regression tests?
- Open questions:
  - Job scheduling: requiring the availability of all nodes of a cluster is not very realistic. Move to per-node scheduling? (and drop Jenkins?)
  - Respective roles of testbed operators and experimenters?

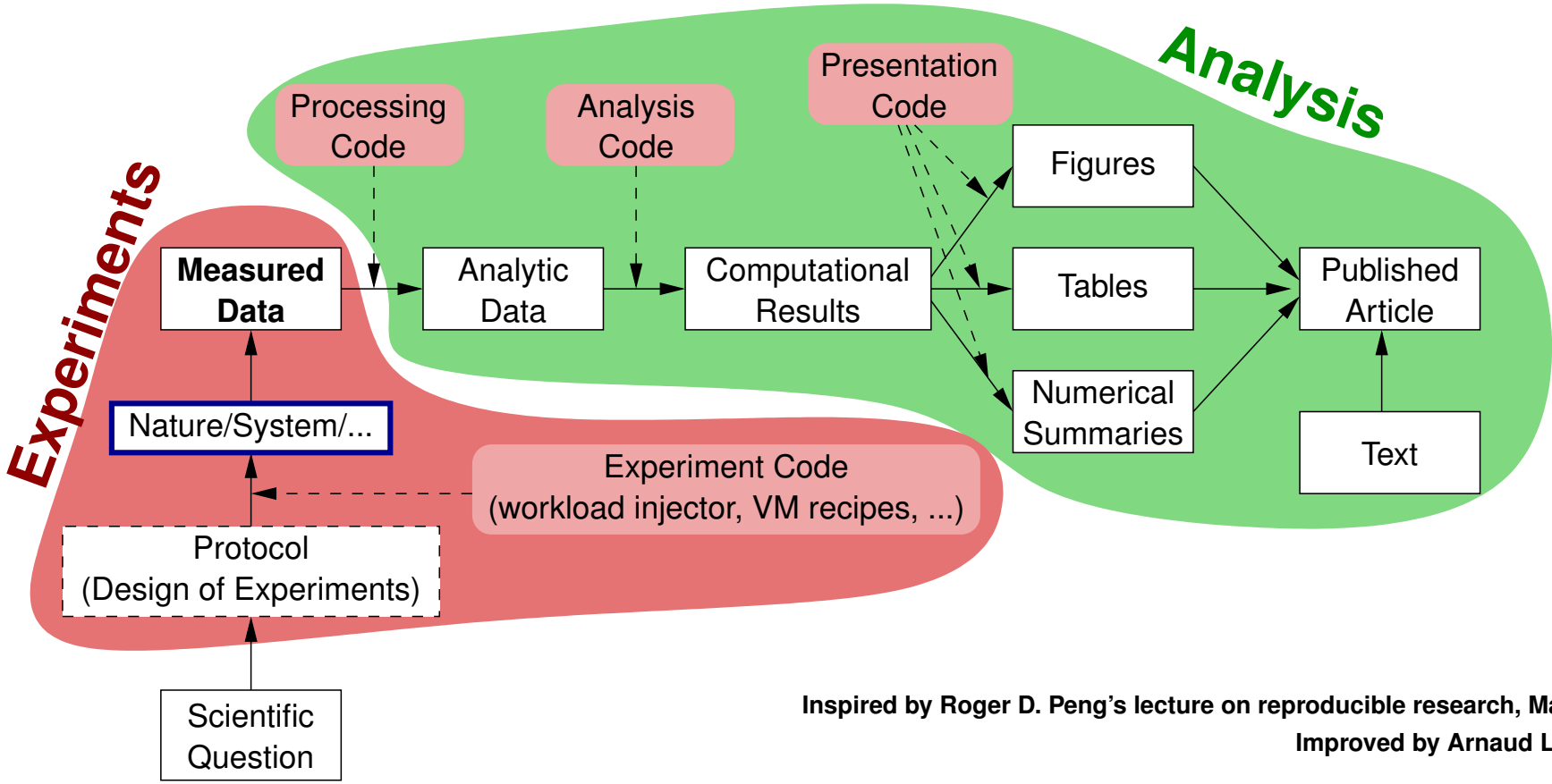
Reproducibility 101



Inspired by Roger D. Peng's lecture on reproducible research, May 2014

Improved by Arnaud Legrand

Reproducibility 101



Inspired by Roger D. Peng's lecture on reproducible research, May 2014
Improved by Arnaud Legrand

How much do you trust your experiments' results?

How much do you trust your simulator or testbed?

Calibration/qualification phase?

- Goal: Make sure that tools and hardware behave as expected
- Challenging task:
  - Many different tools (experiment orchestration solution, load injection, measurement tools, etc.)
  - Mixed with complex hardware, deployed at scale
- Result: very few experimenters do that in practice
  - Most experimenters trust what is provided
- Shouldn't this be the responsibility of the tools maintainers (simulators developers, testbeds maintainers)?

Related work

- Infrastructure monitoring
  - Nagios-like (basic checks to make sure that each service is available)
  - Move to more complex checks (functionality-based) and alerting based on time-series, e.g. with Prometheus (esp. useful on large-scale elastic infrastructures)
- Infrastructure testing
  - Netflix Chaos Monkey
- Testbed testing
  - Fed4FIRE monitoring: `https://fedmon.fed4fire.eu`
    - Check that login, API, very basic usage work
  - Grid'5000 g5k-checks (per-node checks)
    - Similar tool on Emulab (CheckNode)
  - Emulab's LinkTest
    - Network characteristics (latency, bandwidth, link loss, routing)

Resources discovery, verification, selection[4]

- Describing resources $\rightsquigarrow$ understand results
  - Covering nodes, network equipment, topology
  - Machine-parsable format (JSON) $\rightsquigarrow$ 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"
  }
],
```


[4] David Margery et al. “Resources Description, Selection, Reservation and Verification on a Large-scale Testbed”. In: *TRIDENTCOM*. 2014.

Resources discovery, verification, selection[4]

- 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


```
"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"
  }
],
```


[4] David Margery et al. “Resources Description, Selection, Reservation and Verification on a Large-scale Testbed”. In: *TRIDENTCOM*. 2014.

Resources discovery, verification, selection[4]

- 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

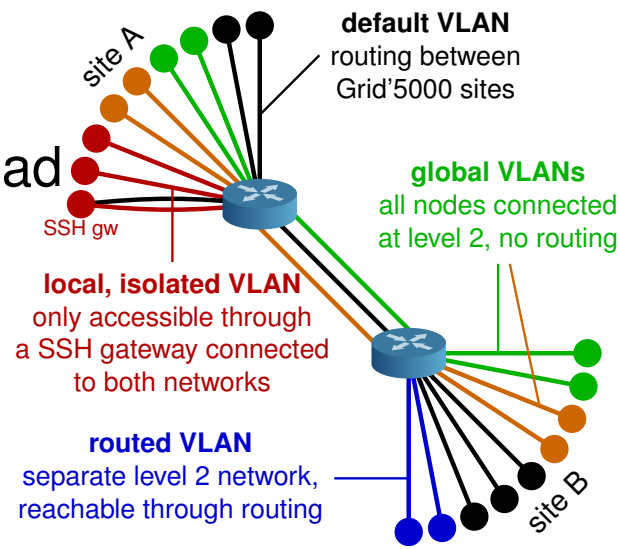

```
"processor": {
  "cache_l2": 8388608,
  "cache_l1": null,
  "model": "Intel Xeon",
  "instruction_set": "",
  "other_description": "",
  "version": "X3440",
  "vendor": "Intel",
  "cache_l1i": null,
  "cache_l1d": null,
  "clock_speed": 2530000000.0
},
"uid": "graphene-1",
"type": "node",
"architecture": {
  "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"
  }
],
```


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

[4] David Margery et al. "Resources Description, Selection, Reservation and Verification on a Large-scale Testbed". In: *TRIDENTCOM*. 2014.

Reconfiguring the testbed

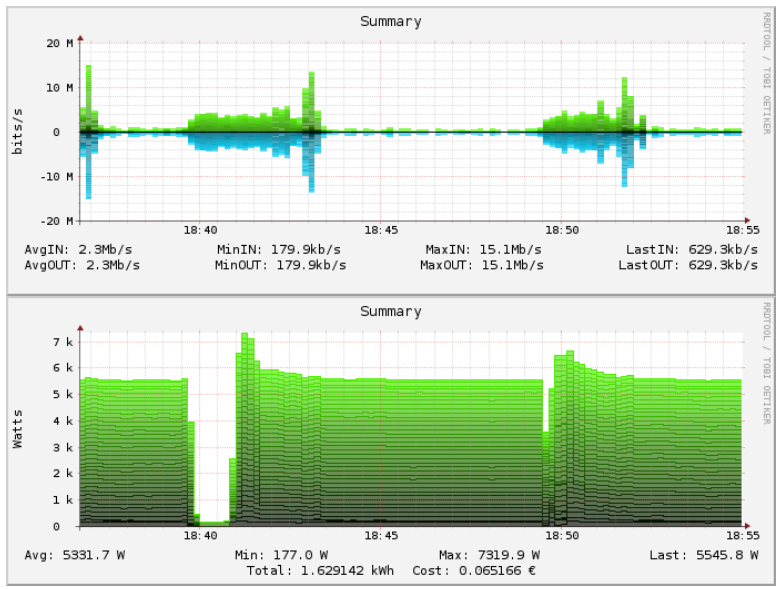
- Operating System reconfiguration with Kadeploy:
  - Provides a *Hardware-as-a-Service* cloud infrastructure
  - Enable users to deploy their own software stack & get *root* access
  - **Scalable, efficient, reliable and flexible**:
    200 nodes deployed in ~5 minutes
  - Images generated using Kameleon for traceability
- 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



Experiment monitoring

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 (≈1 Hz)
  - Live visualization
  - REST API
  - Long-term storage



Why Jenkins, after all?

- Several Jenkins limitations were worked-around
- Was using Jenkins really a good choice in the first place?
- Yes. Benefits:
  - Clean execution environment for scripts
  - Queue to control overloading
  - Access control for users to trigger jobs manually with a web interface
  - Long-term storage of results history and test logs
- (Also, our Jenkins instance is increasingly used for traditional CI/CD talks)