A Unified Monitoring Framework for Energy Consumption and Network Traffic

Florentin Clouet[1,2,3], Simon Delamare[4,5,6], Jean-Patrick Gelas[4,5,6], Laurent Lefèvre[4,5,6], Lucas Nussbaum[1,2,3], Clément Parisot[1,2,3], Laurent Pouilloux[4,5,6], François Rossigneux[4,5,6]

[1]Inria, Villers-lès-Nancy, France
[2]Université de Lorraine, LORIA, France
[3]CNRS, LORIA - UMR 7503, France
[4]Inria, Montbonnot-Saint-Martin, France
[5]Université de Lyon, LIP, France
[6]CNRS, LIP - UMR 5668, France

firstname.lastname@inria.fr

ABSTRACT

Providing experimenters with deep insight about the effects of their experiments is a central feature of testbeds. In this paper, we describe Kwapi, a framework designed in the context of the Grid'5000 testbed, that unifies measurements for both energy consumption and network traffic. Because all measurements are taken at the infrastructure level (using sensors in power and network equipment), using this framework has no dependencies on the experiments themselves. Initially designed for OpenStack infrastructures, the Kwapi framework allows monitoring and reporting of energy consumption of distributed platforms. In this article, we present the extension of Kwapi to network monitoring, and outline how we overcame several challenges: scaling to a testbed the size of Grid'5000 while still providing high-frequency measurements; providing long-term loss-less storage of measurements; handling operational issues when deploying such a tool on a real infrastructure.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*; B.8.2 [**Performance and Reliability**]: Performance Analysis and Design Aids

General Terms

Experimentation, Measurement, Performance

Keywords

Experimentation, monitoring, measurement, energy consumption, network traffic

1. INTRODUCTION

State-of-the-art experimental research requires testbeds with a number of features and services. The most important one is probably the ability to provide experimenters with a very deep control of how their experiments are performed. In the context of distributed systems research, this typically means providing services to control the software stack installed on nodes, to control the placement of nodes for a given experiment and to mitigate interactions with other experiments (congestion and over-provisioning at the networking or virtualization layers), and to isolate one experiment from others to avoid undesired perturbations.

However, another key feature in an experimental environment is the ability to provide *observability* to experimenters, through appropriate monitoring and measurement solutions. This is important to understand performance limitations and bottlenecks during experiments – both those caused by the experiment itself, and those caused by limitations of the testbed (e.g. network architecture) or other concurrent experiments. It is also a prerequisite to evaluate the impact of various solutions on the environment (e.g. network-efficiency, power-efficiency of applications), and to experiment on applications that try to optimize various criteria using feedback loops.

Sometimes measurements can be performed by the experimenters themselves, by instrumenting the application, or through the use of system-level instrumentation and monitoring tools such as DTrace [7], systemtap [14] or Linux's perf tools [12]. But this approach has three main limitations. First, the measurement infrastructure is part of the experiment itself, and might cause a non-negligible impact on the experiment execution and results, depending on how the measurements and the instrumentation are performed. Second, it puts the burden of instrumentation on the experimenters. This can be justified when the instrumentation needs are somehow specific to the experiment, but there are many cases where fairly standard measurements need to be collected. Finally, it cannot provide access to data that is not normally accessible by the experimenter without cooperation from the testbed operators.

Grid'5000 [5] is a testbed for experiment-driven research on

TRIDENTCOM 2015, June 24-25, Vancouver, Canada

DOI 10.4108/icst.tridentcom.2015.259704

HPC, Clouds and Big Data. It is structured in 10 sites (geographical locations) connected together with a dedicated 10-Gbps backbone network. Overall, the testbed contains 1000 nodes (or 8000 CPU cores). The testbed already provides a number of advanced features: resources are automatically described and verified using a *Reference API* [19], experimenters can reconfigure the software stack using Kadeploy [18] and isolate their experiments at the physical network level with KaVLAN [5] in order to execute complex middlewares [4]. However, until recently, Grid'5000 did not provide a monitoring and measurement framework suited to experimenters' needs.

In this paper, we describe Kwapi, the monitoring and measurement framework for energy consumption and network traffic designed for the Grid'5000 testbed. Kwapi relies on measurements taken at the infrastructure level, at a high frequency (one measurement every few seconds), scales to the size of Grid'5000 thanks to a federated architecture, and also provides long-term loss-less storage of measurements.

This paper is organized as follows. Section 2 provides an overview of monitoring and measurement services in the context of CS research infrastructures. Then, Section 3 describes the design of the Kwapi framework, and Section 4 provides some examples of its use and an evaluation of its capabilities. Finally, Section 5 draws some concluding remarks and discusses some possible future work.

2. RELATED WORK

There are a number of motivations for doing monitoring. The most widespread use of those techniques is probably by systems administrators, in order to ensure that an infrastructure is functioning correctly and understand trends in resource utilization. Historically, the common ancestor to most of today's solutions is MRTG – the Multi Router Traffic Grapher [22]. Initially designed to monitor network devices over the Simple Network Management Protocol[9] (SNMP), it was largely extended through external plugins to monitor other kinds of services. The data storage, logging and graphing component of MRTG evolved into RRDtool [21], which is used as a basis for most of today's standard monitoring solutions such as Cacti [1], Munin [3] or Collectd [2]. Those solutions meet the needs of systems administrators to monitor all kinds of devices, systems and services, up to coffee machines [25]. More specialized solutions address more specific problems, such as Ganglia [20], that uses a distributed architecture to monitor large-scale HPC clusters.

While those monitoring solutions are typically used by testbed operators to monitor the research infrastructure itself, they are unsuitable for most experimenters. First, while system administrators typically need to identify long-term tendencies (e.g. average network usage slowly increasing to the physical limits), experimenters typically require a much higher measurement frequency in order to get deep insight into what happens during their experiments and identify short phenomena. Off-the-shelf monitoring solutions are not designed, and cannot be configured, to perform measurements at high frequency. Most of them choose a default measurement interval of 5 minutes. Second, in the context of experimentation, it is useful to integrate long-term data storage in the monitoring system itself. The use of RRDTool as a basis for long-term data storage has major limitations, as it is designed to support archival of old data by averaging the data over time periods. While it makes a lot of sense in the system administration context, it causes a loss of precision if measurements need to be extracted long after they have been taken.

Some testbeds include monitoring services. PlanetLab provided CoMon [23] (now discontinued) to gather information on the status of nodes and slices. It could be used to see what is affecting the performance of nodes, and to examine the resource profiles of individual experiments, with an update frequency of 5 minutes. Its goals were to help users identify problematic nodes (e.g. overloaded) and select appropriate resources for their experiments. Another PlanetLab service, PlanetFlow [17], provides accountability of network traffic inside the testbed, by monitoring all network traffic and link them to slices and users, in order to enable system administrators to react appropriately to complaints about traffic originating from PlanetLab. More efficient strategies to that kind of flow monitoring have also been explored in Pegasus [15].

In terms of overall objectives, the work that is the closer to the one presented in this paper is the ORBIT Measurement framework and Library (OML) [26], which was designed in the context of the ORBIT testbed and is actively used on other OMF-based testbeds. OML is an instrumentation and monitoring framework. In handles the collection of various kinds of measurements from applications and services, and providers filters so that experimenters can select precisely the measurements needed for their experiments.

Kwapi, the monitoring service presented in this paper, covers both network traffic and energy consumption. While the works mentioned previously in this section often cover the monitoring of network traffic, there are very few attempts at organizing the monitoring of energy consumption. One can however mention the previous iteration of Kwapi [24], that was integrated with the OpenStack cloud platform, and the former works [10, 11] of the Kwapi designers that inspired Kwapi's design choices.

3. THE KWAPI FRAMEWORK

This section introduces Kwapi, our monitoring tool for energy consumption and network traffic designed in the context of the Grid'5000 testbed. After an overview of the framework, we will detail implementation and deployment challenges that arose during this development.

3.1 Overview

The Kwapi framework was originally developed to provide power consumption measurements to the cloud computing software platform OpenStack, in the context of the XLCloud project[1].

Its architecture (Figure 1) is based on a layer of drivers, which retrieve measurements from several devices, and a layer of plugins that collect and process them. The communication between these two layers goes through a bus,

[1] `http://www.xlcloud.org/`

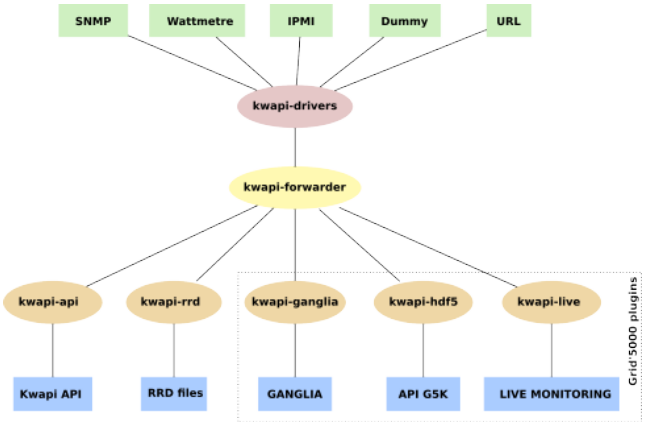

Figure 1: Architecture of Kwapi

the forwarder.

The driver layer is controlled by a Driver Manager that reads the configuration file and initializes a thread for every entry found. An entry consists in the list of probes, the kind of driver to use and the relevant parameter (e.g. SNMP OID). Each driver retrieves measurements and pushes them to the forwarder in JSON format, using ZeroMQ as transport layer.

The Forwarder is an internal communication bus based on ZeroMQ. It works using a publish/subscribe pattern where the drivers are publishers and the plugins are subscribers. It can work locally, i.e. with publishers and subscribers on the same machine, or through a gateway machine to connect isolated networks. In the case of a distributed architecture, a plugin can listen to several drivers located at remote locations using the forwarder.

A plugin is a data consumer that retrieves and processes measurements from the Forwarder. In the original Kwapi framework, only two plugins were present: a REST API that allows an external system to access real-time measurements, and a visualization plugin based on Round-Robin Database files that expose metrics through a web interface.

3.2 Extending Kwapi

To adapt it to the context of experimentation testbeds and Grid'5000, several new features were developed in Kwapi.

Independence from OpenStack

The Kwapi framework started as a contribution to OpenStack and was tightly linked to the OpenStack common libraries (Oslo libraries). For example, these libraries were used to authenticate a Kwapi instance against the Keystone identity service. In order to use Kwapi in a non-OpenStack context this dependency was made optional.

Multi-metrics support

The original purpose of Kwapi, a.k.a. KiloWattAPI, was to provide a scalable and distributed approach to electrical energy monitoring. However, this approach applies to any kind of metrological data, especially network traffic that can be retrieved via SNMP requests on network equipment. As a consequence, we changed several components of Kwapi (driver manager, plugins) to be able to store multiple kinds of metrics.

New drivers

The modularity of the driver plugin is a strong advantage for the framework extension capacity. Namely, one can create a driver that retrieves new kinds of data and publishes them to the forwarder. In the context of Grid'5000 testbed, we developed a new driver, *JSON_URL*, that retrieves metrics from a remote machine. We also changed the SNMP driver to handle *Counter* type metrics used for network traffic.

Improving visualization

The visualization plugin used Round Robin Database (RRD) files to produce PNG images of the power consumption and offer a representation of the metric. The visualization part was separated from the RRD plugin to create a new plugin that takes advantage of Grid'5000 infrastructure for probe selection (i.e. use a batch job number to get the relevant probes).

Long-term storage plugin

As stated before, experimenters take advantage of fine grained data from various metrics to better understand their experiments (i.e., identify bottlenecks, find issues in execution). To deal with this requirement, we added a plugin to store data in HDF5 file format. More details on this plugin will be given in Section 3.3.3.

Integration with existing monitoring systems

In Grid'50000, a Ganglia monitoring system was already deployed to collect various kinds of metrics. These metrics are then exposed to users through the Grid'5000 metrology API. Thanks to Kwapi's modularity, it only took a few days to create a new plugin that pushes data to this pre-existing monitoring infrastructure. As a consequence, all metrics collected by Kwapi are available to Grid'5000 users through the same API that was previously available.

3.3 Deployment challenges in Grid'5000

Designing a high frequency monitoring infrastructure for large scale platform raises various challenges. This section will present the main issues found during Kwapi's deployment on Grid'5000, and what solutions have been implemented to address them.

3.3.1 Scaling the deployment to the size of the testbed

Ensuring scalability is critical for monitoring over large scale platform such as the Grid'5000 testbed. Grid'5000 provides more than 1000 nodes for users' experiments. Each of these nodes has several metrics to be monitored. Therefore, the total number of information to collect is high and grows linearly with the number of nodes. Managing such large amount of data requires to carefully design monitoring tools and platform infrastructure to ensure proper functioning.

As shown in Figure 2, Grid'5000 is distributed over 10 sites. From its origin, the testbed as been designed to be resilient: each site is independent and is able to operate by itself. This has several advantages: for instance, in case of backbone network outage, experiments inside Grid'5000 sites do not fail.

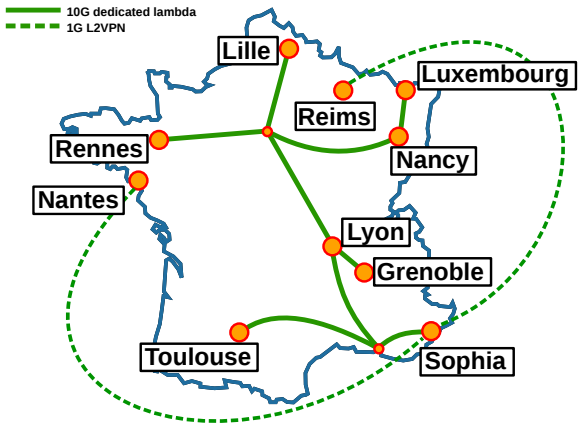

Figure 2: Map of the 10 sites of the Grid'5000 testbed

In addition, it helps scalability: as every services are replicated on each site, their loads are distributed, and adding a new site to Grid'5000 does not interfere with others. Such distributed infrastructure benefits to Kwapi's scalability: individual instances of Kwapi are deployed on each Grid'5000 site and the monitoring workload is shared among them.

3.3.2 *Networking challenges*

Network communications needed to fetch the monitoring data grow with the number of metrics to be collected and the frequency of measurements. On a large testbed and a fine-grained monitoring solution, the amount of traffic might be significant. To ensure accuracy of experimentation results on the testbed, it is essential that this traffic does not interfere with users' experiments traffic. On Grid'5000, two distinct networks are available, provided by different devices and VLANs: one is dedicated to users experiments, the other to communications required to operate the platform. Monitoring traffic induced by Kwapi uses the latter, and therefore has no effect on the experiments' traffic.

As the number of network requests needed to fetch information from monitoring devices is high, it is needed to minimize their number, while ensuring collecting all metrics at the desired frequency. Kwapi implements this strategy by using various optimizations: for instance, it will not make one SNMP request for each metric to collect, but use the *GetBulkRequest* SNMP message[9] on a monitoring device to fetch several metrics at once. The total number of network requests is thus decreased, as well as the workload on monitoring devices needed to handle them.

Other network-related issues has been addressed during the development of Kwapi. For example, depending on the network device vendor or model, SNMP implementation might slightly differ. Network counters are not refreshed at the same frequency: Kwapi typically fetches and stores data every seconds, but it has been adapted to handle lower frequencies and cope with less precise hardware. In addition, older implementations of SNMP used a 32 bits counter to store the total number of octets received or sent on an interface. On a 10 Gbps network, less than 4 seconds are needed to cycle this kind of counters. Hopefully most modern hardware provide 64 bits version of these counters, which Kwapi uses.

3.3.3 *Long-term storage of monitoring data*

In the context of large scale experimentation testbed, storage of monitoring information raises specific issues. In particular, data must be stored in a loss-less way to ensure that complete monitoring information remains available after an experiment has ended. This enables *a posteriori* understanding of experimentation results, and encourage experimenters not to use their own tool to collect data during their experiment. This prevents using a RRD-based solution, as commonly used by monitoring solution, and led us to choose to store monitoring data inside HDF5 files.

In addition, as for network communication, data storage requirements increase with the number of metrics and the frequency of measurements. In a large deployment such as Grid'5000, it is larger than usual and the storage solution must be carefully designed. The typical storage space required is 20 MB per metric every month. On Grid'5000, where three metrics (power consumption, network traffic input and output) are monitored on each of the 1000 nodes, the approximate required size to keep one year of monitoring data is 720 GB. The loss-less data storing and limited capacity available for storage imposes to retain monitoring data during a bounded period of time. Monitoring data oldest than this boundary must be deleted, ensuring a constant storage space usage.

Contrary to RRD, the HDF5 format does not implement any data rotation mechanism. A similar feature has been implemented inside Kwapi's HDF5 plugin as ability to split and merge HDF5 files: every month (this period is configurable), Kwapi uses a new HDF5 file to store monitoring data. Files are kept during one year, ensuring a constant storage space occupation, but the storage duration could be extended if useful. Then, Kwapi is able to retrieve the proper monitoring among the HDF5 files depending on request content it receives, even if the age of requested data span several files.

Other minor issues have been encountered during the HDF5 plugin implementation: a synchronization mechanism has been developed to enable concurrent read/write access to HDF5 files, and data caching has been bypassed to avoid memory leaks caused by the high input rate of data to manage.

3.3.4 *Automated configuration*

The Grid'5000 testbed, as a large scale platform, constantly evolves: transient failures occurs, new hardware devices are added and older ones are removed. When such evolution involves monitoring devices, monitoring tools must be updated to take into account those modifications. However, it would be impractical for platform operators to manually update configurations each time the platform change.

To address this situation, a Kwapi plugin was developed to handle the automated configuration of Kwapi based on the resources description provided by the Grid'50000 Reference API[19]. This plugin is able to configure both network and energy monitoring metrics. Therefore, any changes on the platform reflected inside the Grid'5000 Reference API will

```
$ curl -k https://api.grid5000.fr/3.0/sites/nancy/
    network_equipments/sgriffon1?pretty

{
  "model": "3com 4500g",
  "mtu": 9216,
  "site": "nancy",
  "snmp_community": "public",
  "type": "network_equipment",
  "uid": "sgriffon1",
  "backplane_bps": 176000000000,
  "kind": "switch",
  "linecards": [
    {
      "kind": "node",
      "snmp_pattern": "GigabitEthernet1/%LINECARD%/%PORT
          %",
      "ports": [
        {
        },
        {
          "uid": "griffon-1"
        },
        {
          "uid": "griffon-2"
        },
        {
          "uid": "griffon-3"
        },
        {
          "uid": "griffon-4"
        },
(...)
    }
    {
      "rate": 10000000000,
      "snmp_pattern": "Ten-GigabitEthernet1/%LINECARD%/%
          PORT%"
      "ports": [
        {
        },
        {
          "kind": "node",
          "port": "eth2",
          "uid": "griffon-11"
        },
        {
          "uid": "gw-nancy"
        }
      ],
    },
(...)
}
```

Figure 3: Description of a Networking Device in Grid'5000 Reference API

lead to the reconfiguration of Kwapi monitoring, without any human intervention.

Kwapi's automatic network monitoring configuration reads the description of each network devices from the Reference API. The API provides a JSON based document describing, for each line-card and ports of a given device, the connected node's *hostname*, as shown in Figure 3. The document also provides a *SNMP pattern entry* in order to generate the SNMP "IF-Descr" from the IF Management Information Base [9]. From this information it is possible for the Kwapi plugin to get any SNMP information related to the device's port, and thus network metrics regarding node connected to it. This solution also allows Kwapi to fetch network metrics for every Grid'5000 nodes in a uniform way, while there is a great variety in Grid'5000's network devices (15 different models from 7 different vendors), and which do not implement SNMP descriptions the same way.

Kwapi's configuration for energy monitoring is also performed automatically thanks to the Grid'5000 Reference API. Energy monitoring devices used in Grid'5000 are manifold [13]: electrical consumption can be provided through SNMP by Power Distribution Units (PDU) that power nodes, or by a dedicated device, plugged on nodes' Power Supply Unit (PSU). Grid'5000 Reference API describes each energy monitoring device as a JSON document, as shown in Figure 4.

```
$ curl -k https://api.grid5000.fr/3.0/sites/nancy/pdus/
    graphene-pdu7?pretty

{
  "type": "pdu",
  "uid": "graphene-pdu7",
  "vendor": "Eaton Corporation",
  "sensors": [
    {
      "power": {
        "per_outlets": true,
        "snmp": {
          "available": true,
          "outlet_prefix_oid": "iso
              .3.6.1.4.1.534.6.6.7.6.5.1.3.0",
          "total_oids": [
            "iso.3.6.1.4.1.534.6.6.7.5.5.1.3.0.1",
            "iso.3.6.1.4.1.534.6.6.7.5.5.1.3.0.2"
          ],
          "unit": "W"
        }
      }
    }
  ],
(...)
}
```

Figure 4: Description of a Energy Monitoring Feature of a Power Distribution Unit in Grid'5000 Reference API

```
$ curl -k https://api.grid5000.fr/3.0/sites/nancy/clusters
    /graphene/nodes/graphene-105?pretty
{
  "type": "node",
  "uid": "graphene-105",
  "architecture": {
    "platform_type": "x86_64",
    "smp_size": 1,
    "smt_size": 4
  },

(...)

  "sensors": {
    "power": {
      "available": true,
      "via": {
        "api": {
          "metric": "pdu"
        },
        "pdu": [
          {
            "port": 1,
            "uid": "graphene-pdu7.nancy.grid5000.fr"
          }
        ]
      }
    }
  },

(...)

}
```

Figure 5: Node Description in Grid'5000 Reference API

This description gives methods to fetch power consumption data, for example by providing the SNMP OID to request. The monitoring device available for a given Grid'5000 node is available through the node description. For example, as shown in Figure 5, description of a node connected to a PDU providing per-outlet energy consumption monitoring will contain the monitoring device *uid* as well as the PDU outlet's *port* to which it is connected. It is thus possible for the Kwapi configuration plugin to retrieve how to collect energy consumption data, using SNMP or any other technology depending on the monitoring device, for every Grid'5000 nodes described in the Reference API.

Energy monitoring on modern hardware is a complex task. One difficulty is introduced by the diversity of nodes' electrical powering systems. The simplest case is a node powered by a single PSU, but commonly, a node has two redundant

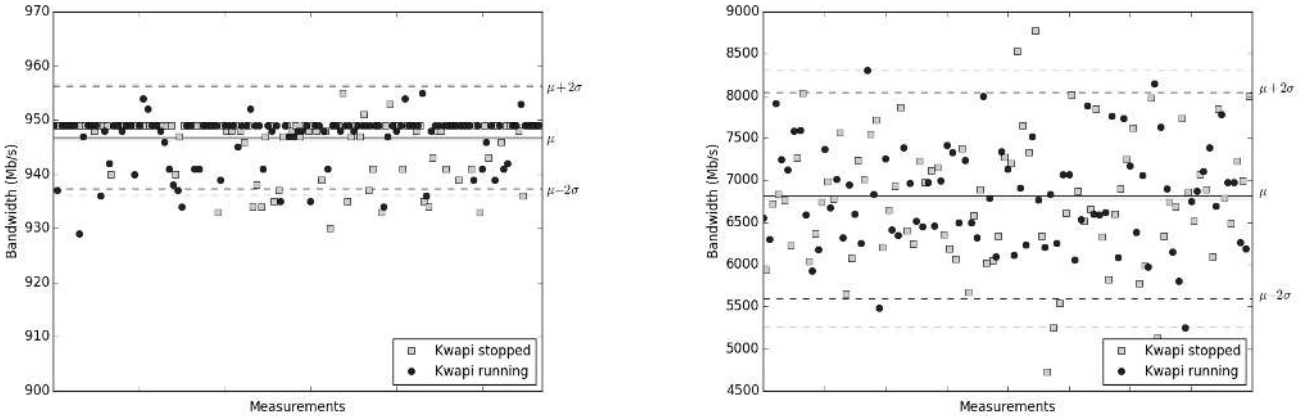

Figure 6: Measurements of network bandwidth between nodes with/without Kwapi running, for a 1G cluster (left panel) and a 10G cluster (right panel). Solid lines represent the mean μ and dashed lines the 2 standard deviations σ from the mean.

PSUs. Both must be monitored and their consumption must be added to compute the node's total consumption. More complex situations involve nodes that share one or more individual PSUs, as for blade servers. Thanks to Grid'5000 Reference API which provides an elaborate description of these various monitoring "topologies"[2], Kwapi's self configuration can automatically handle all cases found on Grid'5000 (9 in total). Through its API and user interface, it also advertises users about these specifics, informing them how energy monitoring data is collected on nodes.

Automated configuration is a key feature for large scale platform monitoring tool. Kwapi's automated configuration plugin that reads the Grid'5000 Reference API to build the list of devices to monitor for every nodes efficiently addresses this challenge. It allows transparent updates on platform hardware changes, and handling of the diversity of monitoring devices used in Grid'5000 in a uniform way.

4. EVALUATION

This section provides information about Kwapi's current status and suitability for experimenters. The typical questions that we try to answer are: (0) Does Kwapi introduce monitoring overheads ? (1) Is Kwapi useful to provide detailed information about phenomena observed during experiments? (2) Is it reactive enough to serve as data input to guide experiments in a feedback loop? (3) Could data from its long-term storage be used to understand trends? (4) Is it ready for production?

4.1 Evaluating Kwapi's monitoring overhead

In order to evaluate the overhead introduced by Kwapi, we perform some measurements on two clusters: *stremi*, a 44-nodes AMD cluster connected via a Cisco C3560E-48TD-S switch; *petitprince*, a 16-nodes Intel cluster connected via Force10 MXL switch. We perform simultaneous measurements using `iperf` in a ring fashion, i.e. we measure simultaneoulsy the bandwidth from node i to node $i+1$. With this setup, we are able to saturate the network equipement and can check if the average bandwidth depends on Kwapi state. For both cases, we perform 5 iterations of the measurements process.

As shown in figure 6, we can see for both clusters that the mean bandwidth is very similar: for *stremi*, we obtain $\mu = 946.35 \pm 5.1$ and $\mu = 946.79 \pm 4.72$ with Kwapi stopped and running respectively ; for *petitprince*, we obtain $\mu = 6781.77 \pm 761$ and $\mu = 6819.13 \pm 612$ respectively. This variations are very little compared to the natural deviations of the iperf measurement process, especially for 10G network equipments.

In addition to these bandwidth estimates, we have also checked that the CPU consumption of the equipment is not affected by the 1-second period SNMP requests that Kwapi perform to retrives the network measures. With this two observations, we can state that our monitoring infrastructure has a very small overhead on the monitored equipments.

4.2 Measuring power consumption while power-cycling machines

Grid'5000 provides experimenters with the ability to turn off and on reserved machines through an interface provided by Kadeploy [18]. Figure 7 shows the network usage and power consumption on all machines of the Reims Grid'5000 site (44 machines) during the following operations:

- 18:39:28 – machines are turned off;
- 18:40:28 – machines are turned on again, and generate network traffic as they boot via PXE;
- 18:49:28 – machine reservation is terminated, causing a reboot to Grid'5000's default system environment.

Those graphs are generated real-time when the experimenter connects to the Kwapi visualization interface. It is also possible to extract this data from the Kwapi API, or to retrieve this data after the experiment using the HDF5 export.

[2]Registered users may see examples on Grid'5000 wiki: `www.grid5000.fr/w/Talk:Reference_Repository`

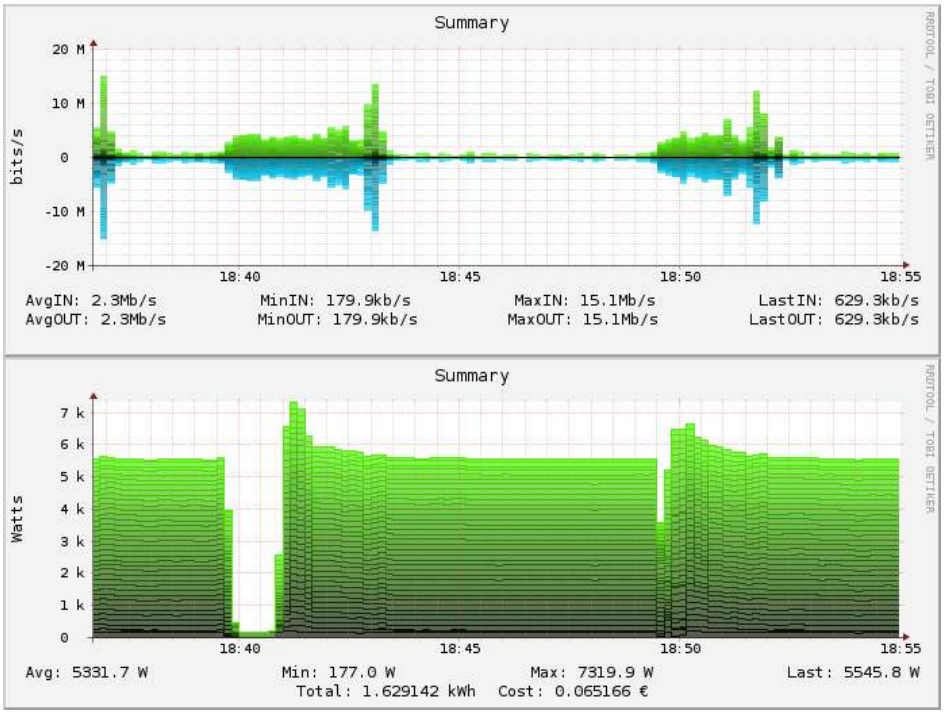

Figure 7: Visualization of network usage (top graph) and power consumption (bottom graph) while machines are turned off and on

4.3 Visualizing TCP congestion control

Given its high frequency of measurement, Kwapi makes it possible to observe short lived phenomena. A good example is TCP's congestion control algorithms, which we illustrate by initiating a data transfer using `nuttcp` between two Grid'5000 sites (Rennes and Nancy). As those sites are connected via a 10-Gbps dedicated backbone, the TCP connection is only limited by the speed of each node's network interface (1 Gbps).

In Figure 8, we compare the bandwidth observed via Kwapi for the start of the TCP connections, with two different configurations of the TCP stack: Hystart enabled, and disabled. Hystart [16] is a heuristic added to Linux's implementation of TCP CUBIC in 2008 that measures the connection's RTT to exit slow start as soon as congestion starts happening, and before packet losses are observed. Unfortunately, the Linux implementation suffered from bugs related to timer precision that resulted in lower performance until they were fixed in March 2011[3]. As a result, and as one can observe in Figure 8, the bandwidth grows much more slowly when Hystart is enabled (with its buggy version, found in Linux 2.6.32).

One could argue that the resulting measurements are not as accurate as what could have been collected using `nuttcp` itself, or using Operating Systems counters. That is true. However, the fact that Kwapi measurements are completely passive and do not require any instrumentation on the sending and receiving nodes is a clear advantage in some situations.

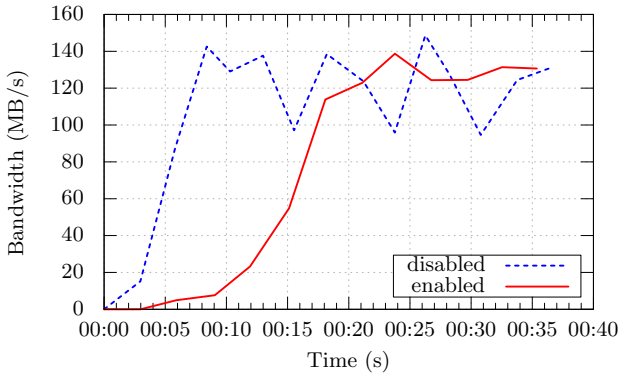

Figure 8: Bandwidth during start of TCP data transfer with/without Hystart

[3] `https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/log/net/ipv4/tcp_cubic.c`

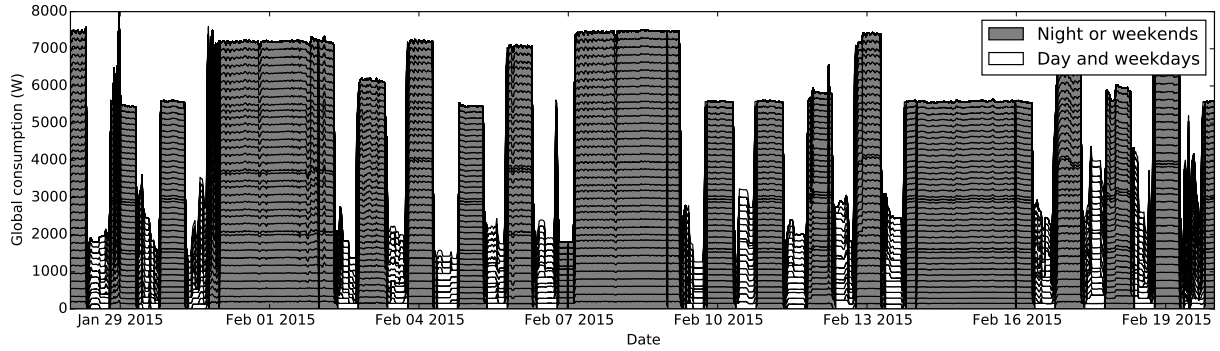

Figure 9: Power consumption of the Reims site. Daytime is in white, while nights and weekends are in gray.

Power consumption (W/node)	Day	Night	Mean
Weekdays	59.44	131.64	101.56
Weekends	153.28	148.41	150.44

Table 1: Consumption trends on Grid'5000 from 2015-01-01 to 2015-02-19

4.4 Extracting long-term power consumption trends

The use of Grid'5000 usage is governed by a set of policies to allow the shared use of the infrastructure by many users. One of these rules distinguish between daytime use of the resources, and night use of the resources. Users must execute large scale jobs during nights (from 19:00pm to 09:00am CET/CEST) and week-ends. During working days (Monday to Friday), users should not use more than the equivalent of 2 hours on all the cores during the given day, to give every user a chance to get some resources to perform small-scale experiments, or do preparatory work for larger-scale experiments.

This section presents an analysis of the power consumption of all 44 nodes from the Reims Grid'5000 site, from 2015-01-01 to 2015-02-19. 1,508,225 power consumption measures were captured over that period, and retrieved from the long-term storage database of Kwapi.

As shown in Table 1, nodes power consumption of the nodes is correlated with the above rule. Weekdays consumption is around 68% lower than weekends. Likewise, the average consumption of a node during the day is 45% lower than the night. Days and nights consumption of nodes during weekends is similar. The same results are also presented in Figure 9, showing again the clear difference between both time periods.

Such results can be useful to predict the Grid'5000 energy cost or to monitor the evolution of consumption on a site. Combined with other statistical measures, Kwapi can offer an overview of the platform usage over long periods of time.

4.5 Checking the Kwapi configuration

The self-configuration functionality offered by the usage of the Grid'5000 API is very powerful but requires post execution verification. In order to validate that data stored is coherent with the real infrastructure, a dedicated tool has been added to the Kwapi framework to perform the following steps: (1) reserve all the nodes on a Grid'5000 site; (2) retrieve the list of probes offered by Kwapi; (3) execute a stress program on the host and check that the evolution of power consumption is as expected; (4) perform some network transfer between a host and a Grid'5000 site frontend and compare with network data provided by Kwapi. It then reports misconfigured probes/hosts to the system administrator.

Similarly to what was done with G5K-checks for machine descriptions in the Reference API [19], this also enables the testbed operators to ensure that the description provided to users is fully accurate.

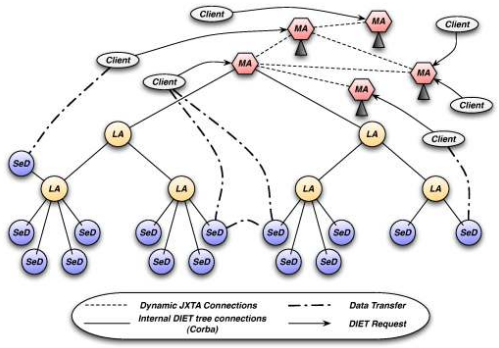

Figure 10: Architecture of DIET middleware

4.6 Evaluating energy-aware schedulers

DIET (Distributed Interactive Engineering Toolbox) [8] is a computing middleware that aims at distributing the scheduling problem across multiple agents. It is able to find appropriate servers according to information given by clients (e.g., problem to be solved, size of the data involved), the performance of the target platform (e.g., server load, available memory, communication performance) and the local availability of data stored during previous computations. The DIET framework is composed of several components as shown in Figure 10.

Recently, energy aware scheduling has been added to DIET[6]: DIET Server Daemons constantly monitor energy consumed by computing resources to process requests submitted by

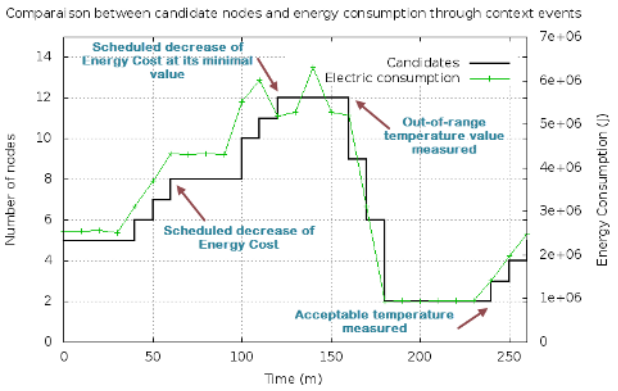

Figure 11: Evolution of DIET's candidate nodes in relation with energy consumption reported by Kwapi

users. DIET builds a knowledge base of energy cost according to various computing resources available (e.g. clusters, computing nodes) and the nature of requests (computing workload, storage requirements, etc.). This information is then used to take energy-aware scheduling decisions.

To help design this feature and validate its implementation, an experiment has been conducted on Grid'5000 that extensively used the Kwapi API. Uniform monitoring over many various computing resources provided by Kwapi deployment on Grid'5000 allowed DIET researchers and developers to focus on scheduling algorithm refinement and validation at a large scale. Figure 11 shows how DIET selects and allocates computing resources according to energy measurements it makes.

4.7 A production-ready code base

Kwapi is publicly available under the Apache 2.0 free software license. Its source code is available on GitHub[4]. It is also available as a Pypi module[5] and as a Debian package to facilitate deployments. On Grid'5000, its installation is fully automated on all sites thanks to the use of Puppet recipes.

5. CONCLUSIONS AND FUTURE WORK

Providing experimenters with deep insight about the effects of their experiments is a central feature of testbeds. We presented Kwapi, a framework designed in the context of the Grid'5000 testbed, that unifies measurements for both energy consumption and network traffic. Kwapi takes all measurements at the infrastructure level (using sensors in power and network equipment), ensuring that the measurement infrastructure has no dependencies on the experiments themselves, and does not cause perturbations to experiments. Kwapi provides long-term, loss-less storage of all measurements in the HDF5 format. Finally, Kwapi was deployed successfully on the Grid'5000 testbed, solving a number of operational challenges caused by the scale of the testbed, and the diversity of equipments.

In the future, this work could be continued in several ways. First, Kwapi could be extended to capture other metrics on the testbed. Low-hanging fruits are additional metrics related to network traffic (e.g. network errors) and energy consumption (e.g. reactive power, in V.A). There are also network technologies popular in the HPC world such as Infiniband that are not supported yet, despite being available on Grid'5000. While they do not provide fine-grained data, sFlow and NetFlow could be supported to provide insight in the type of traffic. Storage systems could also be monitored, which would be useful to understand the behavior of Big Data experiments. Monitoring the temperature at various points of the server rooms could also be relevant for experiments doing power-aware computing (e.g. placing computations on colder machines to balance the room temperature and reduce air cooling usage). Finally, one could envision adding drivers for off-the-shelf monitoring solutions like Collectd or Munin, to benefit from Kwapi's visualization and long term storage features.

The architecture of Kwapi could also be modified. Graphs for the visualization plugin are currently rendered on the server side and transferred over the network, which can be a problem when they are very frequently updated over slow network connections. Client-side generation could be used instead to mitigate this problem. Finally, it would be interesting to bridge Kwapi with other popular monitoring systems, by exporting OML measurements points for Kwapi probes.

[4]`https://github.com/lpouillo/kwapi-g5k`
[5]`https://pypi.python.org/pypi/kwapi-g5k`

6. ACKNOWLEDGMENTS

Part of research on Kwapi is supported by the French FSN (Fonds national pour la Société Numérique) XLcloud project. Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see `https://www.grid5000.fr`).

7. REFERENCES

[1] Cacti – the complete rrdtool-based graphing solution. `http://www.cacti.net/`.

[2] Collectd – the system statistics collection daemon. `https://collectd.org/`.

[3] Munin. `http://munin-monitoring.org/`.

[4] S. Badia, A. Carpen-Amarie, A. Lèbre, and L. Nussbaum. Enabling Large-Scale Testing of IaaS Cloud Platforms on the Grid'5000 Testbed. In *TTC - 1st International Workshop on Testing The Cloud, co-located with ISSTA 2013*, TTC 2013: Proceedings of the 2013 International Workshop on Testing the Cloud, pages 7–12, Lugano, Switzerland, July 2013. ACM - SGIPLAN, ACM.

[5] D. Balouek, A. Carpen Amarie, G. Charrier, F. Desprez, E. Jeannot, E. Jeanvoine, A. Lèbre, D. Margery, N. Niclausse, L. Nussbaum, O. Richard, C. Pérez, F. Quesnel, C. Rohr, and L. Sarzyniec. Adding virtualization capabilities to the Grid'5000 testbed. In I. Ivanov, M. Sinderen, F. Leymann, and T. Shan, editors, *Cloud Computing and Services Science*, volume 367 of *Communications in Computer and Information Science*, pages 3–20. Springer International Publishing, 2013.

[6] D. Balouek-Thomert, E. Caron, and L. Lefevre. Energy-aware server provisioning by introducing middleware-level dynamic green scheduling. In *Workshop HPPAC'15. High-Performance, Power-Aware Computing*, Hyderabad, India, May 2015. In conjunction with IPDPS 2015. To appear.

[7] B. M. Cantrill, M. W. Shapiro, and A. H. Leventhal. Dynamic instrumentation of production systems. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '04, pages 2–2, Berkeley, CA, USA, 2004. USENIX Association.

[8] E. Caron and F. Desprez. Diet: A scalable toolbox to build network enabled servers on the grid. *International Journal of High Performance Computing Applications*, 20(3):335–352, 2006.

[9] J. Case, M. Fedor, M. Schoffstall, and C. Davin. A simple network management protocol (snmp), 1989.

[10] G. Da Costa, M. D. de Assunção, J.-P. Gelas, Y. Georgiou, L. Lefèvre, A.-C. Orgerie, J.-M. Pierson, O. Richard, and A. Sayah. Multi-facet approach to reduce energy consumption in clouds and grids: The green-net framework. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, e-Energy '10, pages 95–104, New York, NY, USA, 2010. ACM.

[11] M. D. De Assuncao, J.-P. Gelas, L. Lefevre, and A.-C. Orgerie. The green grid'5000: Instrumenting and using a grid with energy sensors. In *Remote Instrumentation for eScience and Related Aspects*, pages 25–42. Springer, 2012.

[12] A. C. de Melo. The new linux'perf'tools. In *Slides from Linux Kongress*, 2010.

[13] M. E. M. Diouri, M. F. Dolz, O. Glück, L. Lefevre, P. Alonso, S. Catalán, R. Mayo, and E. S. Quintana-Ortí. Solving some mysteries in power monitoring of servers: Take care of your wattmeters! In *Energy Efficiency in Large Scale Distributed Systems*, pages 3–18. Springer, 2013.

[14] F. C. EIgler, V. Prasad, W. Cohen, H. Nguyen, M. Hunt, J. Keniston, and B. Chen. Architecture of systemtap: a linux trace/probe tool, 2005.

[15] S. Gangam, P. Sharma, and S. Fahmy. Pegasus: Precision hunting for icebergs and anomalies in network flows. In *INFOCOM*, pages 1420–1428, 2013.

[16] S. Ha and I. Rhee. Taming the elephants: New tcp slow start. *Computer Networks*, 55(9):2092–2110, 2011.

[17] M. Huang, A. Bavier, and L. Peterson. Planetflow: Maintaining accountability for network services. *SIGOPS Oper. Syst. Rev.*, 40(1):89–94, Jan. 2006.

[18] E. Jeanvoine, L. Sarzyniec, and L. Nussbaum. Kadeploy3: Efficient and Scalable Operating System Provisioning for Clusters. *USENIX ;login:*, 38(1):38–44, Feb. 2013.

[19] D. Margery, E. Morel, L. Nussbaum, O. Richard, and C. Rohr. Resources Description, Selection, Reservation and Verification on a Large-scale Testbed. In *TRIDENTCOM - 9th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities*, Guangzhou, China, May 2014.

[20] M. L. Massie, B. N. Chun, and D. E. Culler. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(5-6):817–840, 2004.

[21] T. Oetiker. Rrdtool: Round robin database tool. `http://oss.oetiker.ch/rrdtool/`, 1999.

[22] T. Oetiker and D. Rand. Mrtg: The multi router traffic grapher. In *LISA – 12th Systems Administration Conference*, volume 98, pages 141–148, 1998.

[23] K. Park and V. S. Pai. Comon: A mostly-scalable monitoring system for planetlab. *SIGOPS Oper. Syst. Rev.*, 40(1):65–74, Jan. 2006.

[24] F. Rossigneux, L. Lefevre, J.-P. Gelas, and M. Dias De Assuncao. A Generic and Extensible Framework for Monitoring Energy Consumption of OpenStack Clouds. In *The 4th IEEE International Conference on Sustainable Computing and Communications (Sustaincom 2014)*, Sydney, Australia, Dec. 2014.

[25] A. Simon and L. Didry. Munin : superviser simplement la machine à café ... enfin une réalité ! In *JRES - 10èmes Journées Réseaux*, 2013. `https://conf-ng.jres.org/2013/planning.html#article_60`.

[26] M. Singh, M. Ott, I. Seskar, and P. Kamat. Orbit measurements framework and library (oml): motivations, implementation and features. In *Testbeds and Research Infrastructures for the Development of Networks and Communities, 2005. Tridentcom 2005. First International Conference on*, pages 146–152, Feb 2005.