A Generic and Extensible Framework for Monitoring Energy Consumption of OpenStack Clouds

François Rossigneux, Jean-Patrick Gelas, Laurent Lefèvre, Marcos Dias de Assunção
Inria Avalon, LIP Laboratory
Ecole Normale Supérieure de Lyon
University of Lyon, France

Abstract—**Although cloud computing has been transformational to the IT industry, it is built on large data centres that often consume massive amounts of electrical power. Efforts have been made to reduce the energy clouds consume, with certain data centres now approaching a Power Usage Effectiveness (PUE) factor of 1.08. While this is an incredible mark, it also means that the IT infrastructure accounts for a large part of the power consumed by a data centre. Hence, means to monitor and analyse how energy is spent have never been so crucial. Such monitoring is required not only for understanding how power is consumed, but also for assessing the impact of energy management policies. In this article, we draw lessons from experience on monitoring large-scale systems and introduce an energy monitoring software framework called KiloWatt API (KWAPI), able to handle OpenStack clouds. The framework — whose architecture is scalable, extensible, and completely integrated into OpenStack — supports several wattmeter devices, multiple measurement formats, and minimises communication overhead.**

I. Introduction

Cloud computing [1] has become a key building block in providing IT resources and services to organisations of all sizes. Among the claimed benefits of clouds, the most appealing derive from economies of scale and often include a pay-as-you-go business model, resource consolidation, elasticity, good availability, and wide geographical coverage. Despite these advantages when compared to other provisioning models, in order to serve customers with the resources and elasticity they need, clouds generally rely on large data centres that consume massive amounts of electrical power [2] [3].

Although some data centres now approach a Power Usage Effectiveness (PUE) factor of 1.08[1], such a mark means that the IT infrastructure is now responsible for a large part of the consumed power. Means to monitor and analyse how energy is spent are crucial to further improvement, but our previous work in this area has demonstrated that monitoring the power consumed by large systems is not always an easy task [4]–[6]. There are multiple power probes available in the market, generally with their own APIs, physical connections, precision, and communication protocols [7]. Moreover, cost related constraints can lead data centre operators to acquire and deploy equipments at multiple stages, or to monitor the power consumption of only part of an infrastructure.

From a cost perspective, monitoring the power consumption of only a small part of deployed equipments is sound, but it prevents one from capturing important nuances of the infrastructure. Previous work has shown that as a computer cluster ages, certain components wear out, while others are replaced, leading to heterogeneous power consumption among nodes that were seemingly homogeneous [8]. The difference between nodes that consume the least power and nodes that consume the most can reach 20% [9], which reinforces the idea that monitoring the consumption of all equipments is required for exploring further improvement in energy efficiency and evaluate the impact of system-wide policies. Monitoring a great number of nodes, however, requires the design of an efficient infrastructure for collecting and processing the power consumption data.

This paper describes the design and architecture of a generic and flexible framework, termed as KiloWatt API (KWAPI), that interfaces with OpenStack to provide it with power consumption information collected from multiple heterogeneous probes. OpenStack is a project that aims to provide ubiquitous open source cloud computing platform and is currently used by many corporations, researchers and global data centres[2]. We describe how KWAPI is integrated into Ceilometer; OpenStack's component conceived to provide a framework to collect a large range of metrics for metering purposes[3]. With the increasing use of Ceilometer as the *de facto* metering tool for OpenStack, we believe that such an integration of a power monitoring framework into OpenStack can be of great value to the research community and practitioners.

The remaining part of this paper is organised as follows. Section II describes background and related work, whereas Section III presents the KWAPI architecture. Section IV discusses experimental results on measuring the throughput of KWAPI, and Section V concludes the paper.

II. Background and Related Work

This section provides an overview of Ceilometer's architecture and describes related work on monitoring power consumption of large-scale computing infrastructure.

A. OpenStack Ceilometer

Ceilometer — whose logical architecture[4] is depicted in Figure 1 — is OpenStack's framework for collecting performance metrics and information on resource consumption. As of writing, it allows for data collection under three methods:

[1]http://gigaom.com/2012/03/26/whose-data-centers-are-more-efficient-facebooks-or-googles/

[2]http://www.openstack.org/user-stories/

[3]https://wiki.openstack.org/wiki/Ceilometer

[4]http://docs.openstack.org/developer/ceilometer/architecture.html

- **Bus listener agent**, which picks events on OpenStack's notification bus and turns them into Ceilometer samples (*e.g.* cumulative type, gauge or delta) that can then be stored into the database or provided to an external system via publishing pipeline.
- **Push agents**, more intrusive, consist in deploying agents on the monitored nodes to push data remotely to be taken by the collector.
- **Polling agents** that poll APIs or other tools to collect information about monitored resources.

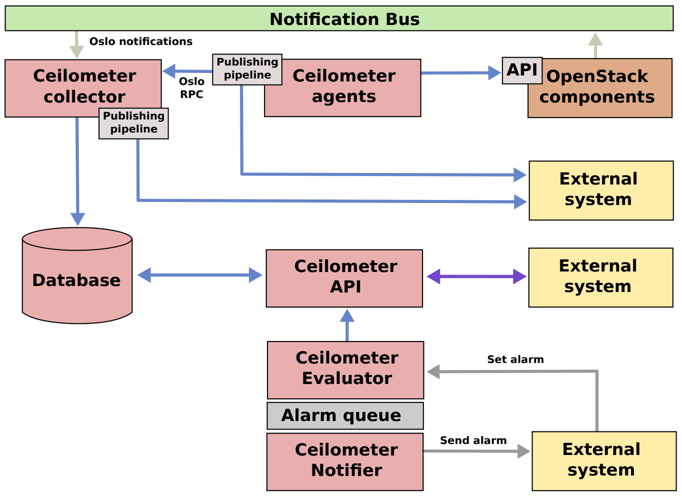


Fig. 1. Overview of Ceilometer's logical architecture.

The last two methods depend on a combination of central agent, computer agents and collector. The compute agents run on nodes and retrieve information about resource usage related to a given virtual machine instance and a resource owner. The central agent, on the other hand, executes *pollsters* on the management server to retrieve data that is not linked to a particular instance. Pollsters are software components executed, for example, to poll resources by using an API or other methods. The Ceilometer database, which can be queried via Ceilometer API, allows an external system to view the history of a resource's metrics, and enables the system to set and receive alarms.

The *hmac* module of Python's library can be used for signing metering messages, and a shared secret value can be provided in the configuration settings. The collector and systems accessing the API use signatures included in the messages for verification.

B. *Energy Monitoring and Efficiency in Clouds*

Over the past years, several techniques have been provided to minimise the energy consumed by computing infrastructure. At the hardware level, for instance, processors are able to operate at multiple frequency and voltage levels, and the operating systems or resource managers can choose the level that matches the current workload [10]. At the resource management level, several approaches are proposed, including resource consolidation [11] and rescheduling requests [4], generally with the goal of switching off unused resources or setting them to low power consumption modes. Attempts have also been made to assess the power consumed by individual applications [12].

A means to monitor the energy consumption is key to assess potential gains of techniques to improve software and cloud resource management systems. Cloud monitoring is not a new topic [13] as tools to monitor computing infrastructure [14], [15] as well as ways to address some of the usual issues of management systems have been introduced [16], [17]. Moreover, several systems for measuring the power consumed by compute clusters have been described in the literature [5]. As traditional system and network monitoring techniques lack the capability to interface with wattmeters, most approaches for measuring energy consumption have been tailored to the needs of projects for which they were conceived.

In our work, we draw lessons from previous approaches to monitor and analyse energy consumption of large-scale distributed systems [4]–[6], [9], [18]. We opt for creating a framework and integrating it with a successful cloud platform (*i.e.* OpenStack), which we believe is of value to the research community and practitioners working on the topic. To the best of our knowledge, this is the first generic energy monitoring framework to be integrated with OpenStack.

III. The KWAPI Architecture

An overview of the KWAPI architecture is presented in Figure 2. The architecture follows a publish/subscribe model based on a set of layers comprising:

- **Drivers**, considered data producers responsible for measuring the power consumption of monitored resources and providing the collected data to consumers via a communication bus; and
- **Data Consumers** — or **Consumers** for short — that subscribe to receive and process the measurement information.

The communication between layers is handled by a bus, as explained in detail later. Data consumers can subscribe to receive information collected by drivers from multiple sites. Both drivers and consumers are easily extensible to support, respectively, several types of wattmeters and provide additional data processing services. A REST API is designed as a data consumer to provide a programming interface for developers and system administrators. In this work it is used to interface with OpenStack by providing the information (*i.e.* by polling monitored devices) required by a *KWAPI Pollster* that feeds Ceilometer.

The following sections provide more details on the main architecture components and their relationship with OpenStack Ceilometer.

A. *Driver Layer*

Drivers are threads initialised by a Driver Manager with a set of parameters loaded from a file compliant with the OpenStack configuration format. These parameters are used to query the meters (*e.g.* IP address and port) and determine the sensor ID to be used in the collected metrics. The measurements that a driver obtains are represented as JavaScript Object Notation (JSON) dictionaries that maintain a small footprint and that

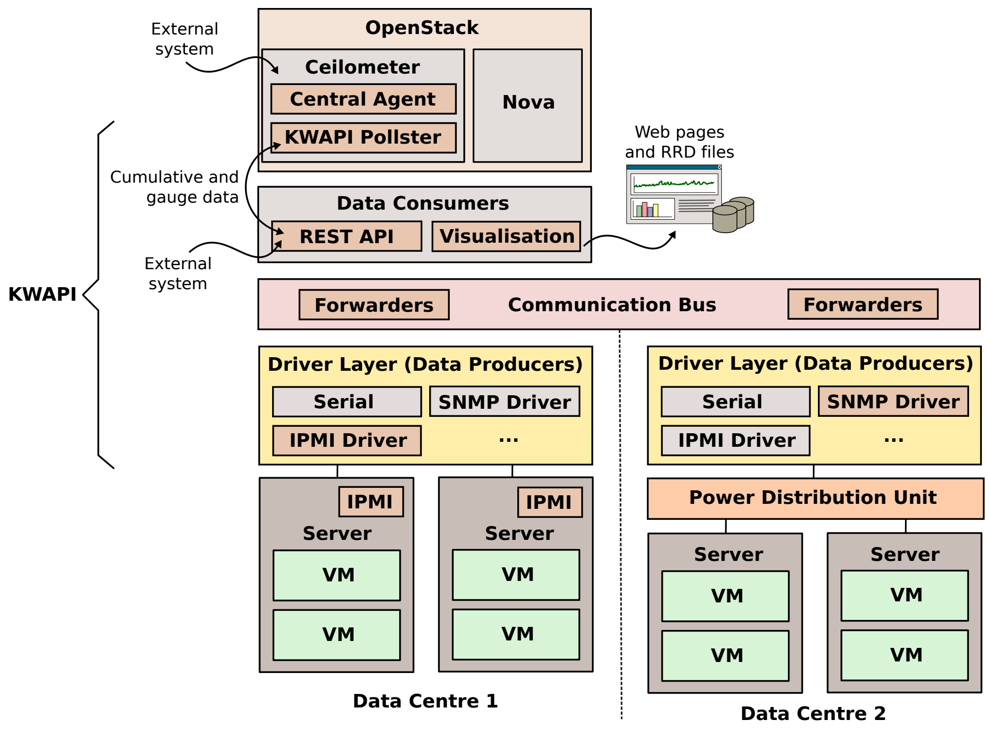


Fig. 2. Overview of KWAPI's architecture.

```
{
    "probe_id": "164",
    "w": 92,
    "a": 0.4,
    "v": 230.0,
    "message_signature": "bb45c88f3ba64..."
}
```

Fig. 3. Example of JSON payload.

can be easily parsed. The size of dictionaries varies depending on the number of fields set by drivers (*i.e.* whether message signing is enabled).

Figure 3 shows a simple example of a JSON payload containing one measurement. Optional fields such as voltage and current can be included. ACK messages have a fixed size of 66 bytes when using TCP connection; drivers and data consumers communicate via IPC sockets when running on the same machine.

Drivers can manage incidents themselves, but the manager also checks periodically if all threads are active, restarting them if necessary. It is important to avoid losing measurements because the reported information is in W instead of kWh. The loss of a measurement may be significant.

Wattmeters available in the market vary in terms of physical interconnection, communication protocols, packaging and precision of measurements they take. They are mostly packaged in multiple outlet power strips called Power Distribution Units (PDUs) or enclosure PDUs (ePDUs), and more recently in the Intelligent Platform Management Interface (IPMI) cards embedded in the computers themselves. Support for several types of wattmeter has been implemented, which drivers can use to interface with a wide range of equipments. In our work, we used IPMI initially at Nova to shutdown and turn on compute nodes, but nowadays we also use it to query a computer chassis remotely.

Although Ethernet is generally used to transport IPMI or SNMP packets over IP, USB and RS-232 serial links are also common. Wattmeters that use Ethernet are generally connected to an administration network (isolated from the data centre main data network). Moreover, wattmeters may differ in the manner they operate; some equipments send measurements to a management node on a regular basis (push mode), whereas others respond to queries (pull mode). Other characteristics that differ across wattmeters include:

- refresh rate (*i.e.* maximum number of measurements per second);
- measurement precision; and
- methodology applied to each measurement (*e.g.* mean of several measurements, instantaneous values, and exponential moving averages).

Table I shows the characteristics of equipments we deployed and used with Kwapi in our cloud infrastructure.

B. Data Consumers

A data consumer retrieves and processes measurements taken by drivers and provided via bus. Consumers expose the information to other services including Ceilometer and visualisation tools. By using a system of prefixes, consumers can subscribe to all producers or a subset of them. When receiving a message, a consumer verifies the signature, extracts the content and processes the data. By default KWAPI provides

TABLE I. WATTMETER INFRASTRUCTURE

Device Name	Interface	Refresh Time (s)	Precision (W)
Dell iDrac6	IPMI / Ethernet	5	7
Eaton	Serial, SNMP via Ethernet	5	1
OmegaWatt	IrDA Serial	1	0.125
Schleifenbauer	SNMP via Ethernet	3	0.1
Watts Up?	Proprietary via USB	1	0.1
ZEZ LMG450	Serial	0.05	0.01

two data consumers, namely the REST API (used to interface with Ceilometer) and a visualisation consumer.

1) REST API: The API consumer computes the number of kWh of each driver probe, adds a timestamp, and stores the last value in watts. If a driver has not provided measurements for a long time, the corresponding data is removed. The REST API allows an external system to retrieve the name of probes, measurements in W or kWh, and timestamps. The API is secured by OpenStack Keystone tokens[5], whereby the consumer needs to ensure the validity of a token before sending a response to the system.

2) Visualisation: The visualisation consumer builds Round-Robin Database (RRD) files from received measurements, and generates graphs that show the energy consumption over a given period, with additional information such as average electricity consumption, minimum and maximum watt values, last value, total energy and cost in Euros. RRD files are of fixed size and store several collections of metrics with different granularities. A web interface displays the generated graphics and a cache mechanism triggers the creation of graphs during queries only if they are out of date. These visualisation resources offer quick feedback to administrators and users during execution of tasks and applications. Figure 4 shows an example of generated graph.

C. Internal Communication Bus

KWAPI uses ZeroMQ [19], a fast broker-less messaging framework written in C++, where transmitters play the role of buffers. ZeroMQ supports a wide range of bus modes, including cross-thread communication, IPC, and TCP. Switching from one mode to another is straightforward. ZeroMQ also provides several design patterns such as publish/subscribe and request/response. As mentioned earlier, in our publish/subscribe architecture drivers are publishers, and data consumers are subscribers. If no data consumer is subscribed to receive data from a given driver, the latter will not send any information through the network.

Moreover, one or more optional forwarders can be installed between drivers and data consumers to minimise network usage. Forwarders are designed to act as especial data consumers who subscribe to receive information from a driver and multicast it to all normal data consumers subscribed to receive the information. Forwarders enable the design of complex topologies and optimisation of network usage when handling data from multiple sites. They can also be used to bypass network isolation problems and perform load balancing.

[5]http://keystone.openstack.org

D. Interface with Ceilometer

We opted for integrating KWAPI with an existing open source cloud platform to ease deployment and use. Leveraging the capabilities offered by OpenStack can help in the adoption of a monitoring system and reduce its learning curve.

Ceilometer's central agent and a dedicated pollster (*i.e.* KWAPI Pollster) are used to publish and store energy metrics into Ceilometer's database. They query the REST API data consumer and publish cumulative (kWh) and gauge (W) counters that are not associated with a particular tenant, since a server can host multiple clients simultaneously.

Depending on the number of monitored devices and the frequency at which measurements are taken, wattmeters can generate a large amount of data, thus demanding storage capacity for further processing and analysis. Management systems often store and perform pre-processing locally on monitored nodes, but such an approach can impact on CPU utilisation and influence the power consumption. In addition, resource managers may switch off idle nodes or set them to stand by mode to save energy, which make them unavailable for processing. Centralised storage, on the other hand, allows for faster data access and processing, but can generate more traffic given that measurements need to be continuously transferred over the network to a central point.

Ceilometer uses its own central database, which is leveraged here to store the energy consumption metrics. In this way, systems that interface with OpenStack's Ceilometer, including Nova, can easily retrieve the data. It is important to notice that, even though Ceilometer provides the notion of a central repository for metrics, it also uses a database abstraction that enables the use of distributed systems such as Apache Hadoop HDFS[6], Apache Cassandra[7], and MongoDB[8].

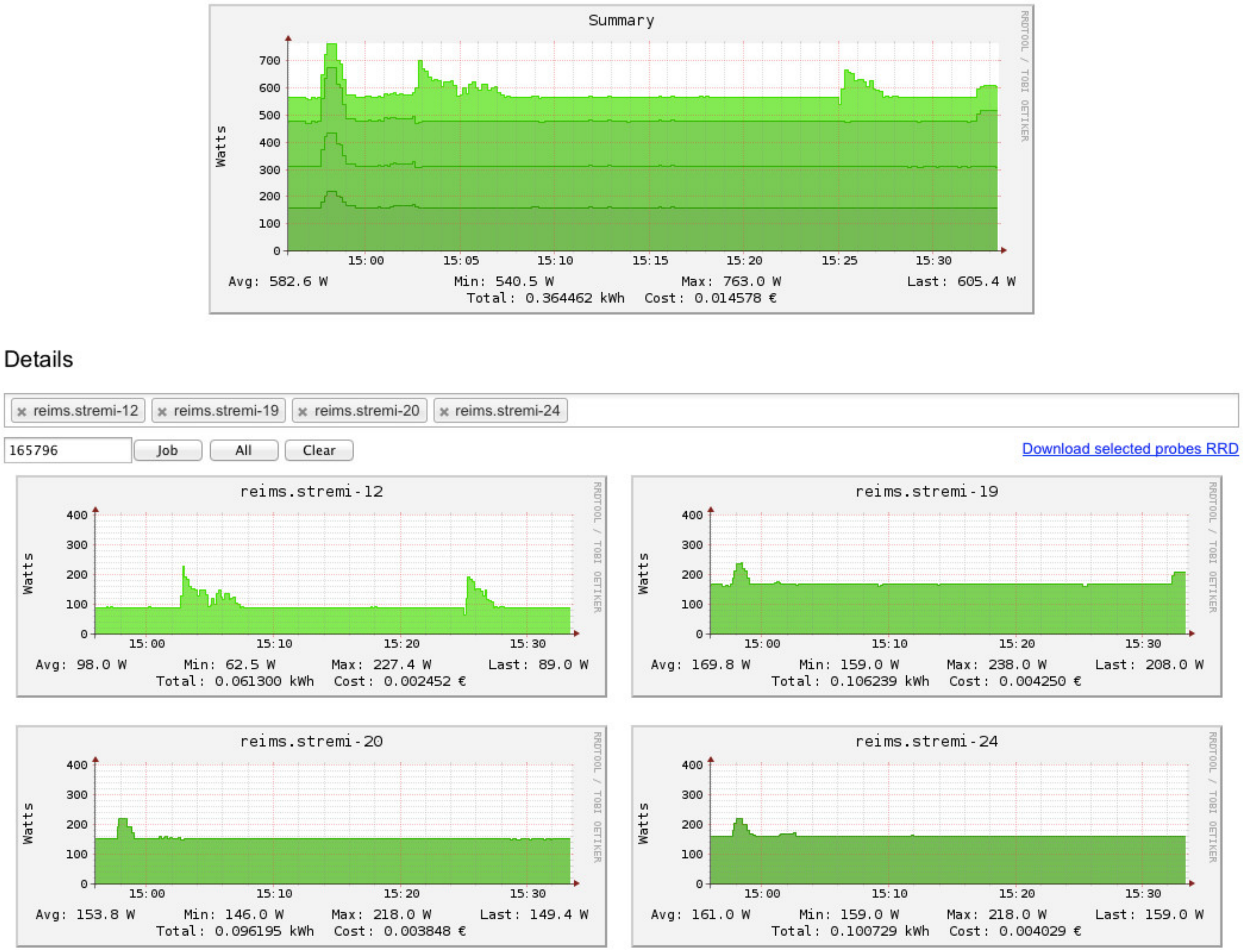
The granularity at which measurements are taken and metrics are computed is another important factor because user needs vary depending on what they wish to evaluate. Taking one or more measurements per second is not common under certain scenarios, which can be a challenge in an infrastructure comprising hundreds or thousands of nodes, demanding efficient and scalable mechanisms for transferring information on power consumption. Hence, in the next section we evaluate the throughput of KWAPI under a few scenarios.

IV. PERFORMANCE EVALUATION

This section provides results of a performance evaluation carried out in our testbed. The goal is not to compare publish/-subscribe systems since such work has already been performed elsewhere [20], [21]. The evaluation demonstrates that the framework serves well the needs of a large range of users of the Grid'5000 platform [22] — the infrastructure we use and where the KWAPI framework is currently deployed in production mode as the means for collecting and providing energy consumption information to users.

First we want to evaluate the CPU and network usage of a typical driver to observe the framework's throughput,

[6]http://hadoop.apache.org/
[7]http://cassandra.apache.org/
[8]http://www.mongodb.org/



Summary — Avg: 582.6 W Min: 540.5 W Max: 763.0 W Last: 605.4 W Total: 0.364462 kWh Cost: 0.014578 €

reims.stremi-12 — Avg: 98.0 W Min: 62.5 W Max: 227.4 W Last: 89.0 W Total: 0.061300 kWh Cost: 0.002452 €

reims.stremi-19 — Avg: 169.8 W Min: 159.0 W Max: 238.0 W Last: 208.0 W Total: 0.106239 kWh Cost: 0.004250 €

reims.stremi-20 — Avg: 153.8 W Min: 146.0 W Max: 218.0 W Last: 149.4 W Total: 0.096195 kWh Cost: 0.003848 €

reims.stremi-24 — Avg: 161.0 W Min: 159.0 W Max: 218.0 W Last: 159.0 W Total: 0.100729 kWh Cost: 0.004029 €

Fig. 4. Example of graph generated by the visualisation plug-in (4 monitored servers).

since provisioning a large number of resources for monitoring purposes is not desirable. For this experiment we deployed the KWAPI drivers and API on a machine with a Core 2 Duo P8770 2.53Ghz processor and 4GB of RAM. We considered:

- a scenario where we emulated 1,000 IPMI cards, each card monitored by a driver thread placing a measurement per second on the communication bus.
- a case with 100 ten-outlet PDUs, each monitored by a driver thread placing ten values per second on the communication bus.

Under both scenarios, 1,000 measurements per second were placed on the bus, even though monitoring was done using different types of probes. We have evaluated these scenarios considering both with and without message signature. Table II summarises the considered scenarios.

Figure 5 shows the results of CPU usage of drivers under the evaluated scenarios. The socket type and number of driver threads do not have a distinguishable impact on the CPU usage. On the test machine, the KWAPI drivers with message signature disabled (*i.e.* IPMI cards unsigned and PDUs unsigned) consumed on average 20% of the total CPU power.

We also evaluated the CPU consumption of the REST API data consumer under the scenarios described in Table II.

TABLE II. SCENARIOS CONSIDERED IN THE EXPERIMENTS.

Scenario name	Agent thread scheme	Message signature
IPMI message signed	1 thread per card	Enabled
IPMI message unsigned	1 thread per card	Disabled
PDU message signed	1 thread per PDU	Enabled
PDU message unsigned	1 thread per PDU	Disabled

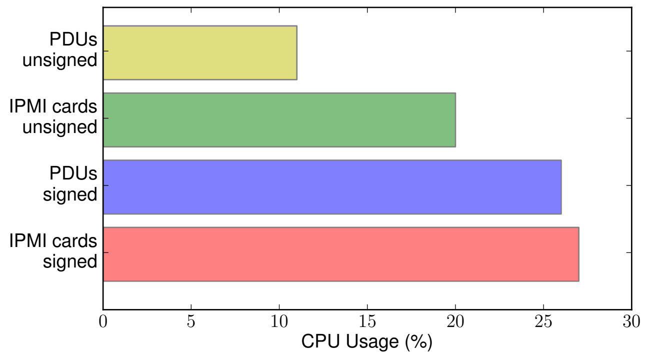


Fig. 5. Driver CPU usage under the evaluated scenarios.

In addition to these scenarios, two conditions were assessed, namely (i) the REST API working as a consumer requesting data from drivers at a one-second time interval (REST API only); and (ii) the API requesting data at one-second interval and also answering a call every second to provide the collected data to an external system (REST API + 1 req/s). Figure 6 summarises the obtained results. The CPU consumption is in general low. Even when message signing is enabled and the API serves a query, its consumption is below 20%. The small variation between the scenarios without message signing is caused by the manner ZeroMQ accumulates data on nodes prior to transmission.

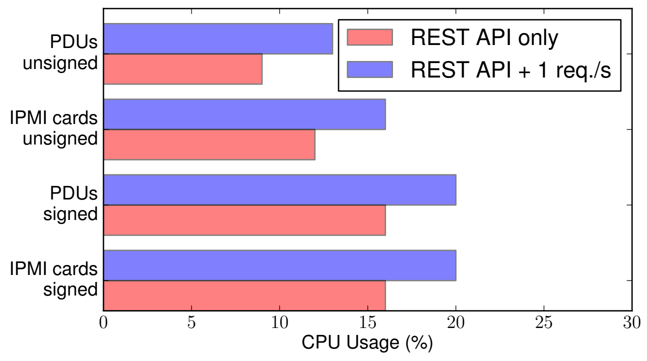


Fig. 6. API consumer CPU usage under the evaluated scenarios.

Although the CPU usage often depends on the drivers, data consumers, and their complexity, and whether message signature is enabled, the experiments show that a large number of probes can be managed by a single machine. In our environment, a management machine per site is more than enough to accommodate the users' monitoring needs. The drivers and API can reuse a machine that already serves other monitoring purposes.

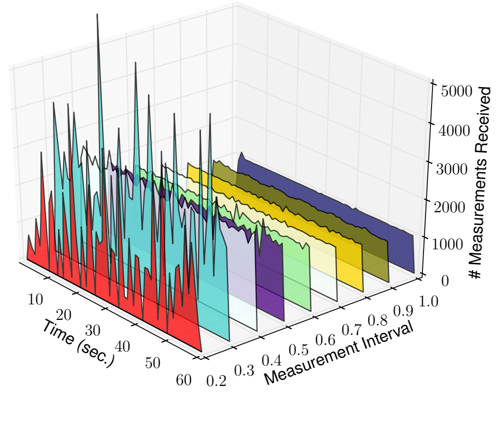


Fig. 7. Number of observations received over a 60 second interval under different multiple intervals.

Although a measurement interval of one second meets the requirements of users in our platform, we wanted to evaluate the impact of using a communication bus in the transfer of observations between drivers and the REST API consumer. In a second experiment we used two machines. On the first machine we instantiated 1,000 driver threads placing random observations on the communication bus. On the second machine we measured the number of measurements that the API is able to receive over a minute. We varied the time between measurements from 0.2 to 1.0 seconds. Figure 7 summarises the obtained results. Though the number of observations generated in this experiment is much higher than what we currently need to handle in our platform, we observe that the framework is able to transfer measurements from drivers to API under a 0.4 second interval without adding much jitter. Under smaller measurement intervals, however, observations start to accumulate and are transferred at large chunks. We believe that under small measurement intervals, and consequently a very large number of observations per second, an architecture based on stream processing systems that guarantees data processing might be more appropriate. Hence, although the framework suits the purposes of large range of users, if measurements are to be taken at very small time intervals, a stream processing architecture would probably yield better performance by enabling the placement of elements to pre-process data closer to where it is generated.

V. Conclusion

In this paper, we described a novel framework (KWAPI) for monitoring the power consumed by resources of an Openstack cloud. Based on lessons learned by monitoring the power consumption of large distributed infrastructure, we proposed an energy monitoring architecture based on a publish/subscribe model. The framework works in tandem with OpenStack's Ceilometer. Experimental results demonstrate that the overhead posed by the monitoring framework is small, allowing us to serve the users' monitoring needs of our large scale infrastructure.

As future work, we intend to explore means to increase the monitoring granularity and the number of measured devices by applying a hierarchy of plug-ins, and a stream processing system with guarantees on data processing [23], [24] for processing streams of measurement tuples.

Acknowledgments

This research is supported by the French Fonds national pour la Société Numérique (FSN) XLCloud project. Some experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the Inria ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://grid5000.fr). Authors wish to thank Julien Danjou for his help during the integration of KWAPI with OpenStack and Ceilometer.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A Berkeley view of Cloud computing," Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, USA, Technical report UCB/EECS-2009-28, February 2009.

[2] J. Baliga, R. W. A. Ayre, K. Hinton, and R. S. Tucker, "Green cloud computing: Balancing energy in processing, storage, and transport," *Proceedings of the IEEE*, vol. 99, no. 1, pp. 149–167, January 2011.

[3] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: Research problems in data center networks," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 1, pp. 68–73, January 2009.

[4] A.-C. Orgerie, L. Lefèvre, and J.-P. Gelas, "Save watts in your Grid: Green strategies for energy-aware framework in large scale distributed systems," in *14th IEEE International Conference on Parallel and Distributed Systems (ICPADS'08)*, Melbourne, Australia, December 2008, pp. 171–178.

[5] M. D. de Assunção, J.-P. Gelas, L. Lefèvre, and A.-C. Orgerie, "The green Grid'5000: Instrumenting and using a Grid with energy sensors," in *5th International Workshop on Distributed Cooperative Laboratories: Instrumenting the Grid (INGRID 2010)*, Poznan, Poland, May 2010.

[6] G. Da Costa, M. D. de Assunção, J.-P. Gelas, Y. Georgiou, L. Lefèvre, A.-C. Orgerie, J.-M. J.-M. Pierson, O. Richard, and A. Sayah, "Multi-facet approach to reduce energy consumption in clouds and grids: The GREEN-NET framework," in *1st International Conference on Energy-Efficient Computing and Networking (e-Energy 2010)*, Passau, Germany, April 2010, pp. 95–104.

[7] M. E. M. Diouri, M. F. Dolz, O. Glück, L. Lefèvre, 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 conference (EE-LSDS)*, Vienne, Autriche, Apr. 2013, pp. 3–18.

[8] J. Dean, "Some potential areas for future research," Google Faculty Summit talk, July 2008.

[9] M. E. M. Diouri, O. Gluck, L. Lefevre, and J.-C. Mignot, "Your cluster is not power homogeneous: Take care when designing green schedulers!" in *International Green Computing Conference (IGCC 2013)*. IEEE, June 2013, pp. 1–10.

[10] G. von Laszewski, L. Wang, A. J. Younge, and X. He, "Power-aware scheduling of virtual machines in DVFS-enabled clusters," in *IEEE International Conference on Cluster Computing and Workshops (Cluster 2009)*, vol. 2, August 2009, pp. 1–10.

[11] A. Beloglazov and R. Buyya, "Openstack neat: A framework for dynamic and energy-efficient consolidation of virtual machines in openstack clouds," *Concurrency and Computation: Practice and Experience*, 2014.

[12] A. Noureddine, "Towards a better understanding of the energy consumption of software systems," PhD thesis, Université Lille 1, Lille, France, March 2006.

[13] G. Aceto, A. Botta, W. de Donato, and A. Pescapè, "Cloud monitoring: A survey," *Computer Networks*, vol. 57, no. 9, pp. 2093–2115, June 2013.

[14] A. Brinkmann, C. Fiehe, A. Litvina, I. Luck, L. Nagel, K. Narayanan, F. Ostermair, and W. Thronicke, "Scalable monitoring system for clouds," in *IEEE/ACM 6th International Conference on Utility and Cloud Computing (UCC 2013)*, December 2013, pp. 351–356.

[15] S. Varrette, V. Plugaru, M. Guzek, X. Besseron, and P. Bouvry, "HPC performance and energy-efficiency of the openstack cloud middlewares," in *Proc. of the 43rd Intl. Conf. on Parallel Processing (ICPP-2014), Heterogeneous and Unconventional Cluster Architectures and Applications Workshop (HUCAA'14)*, September 2014.

[16] J. S. Ward and A. Barker, "Varanus: In situ monitoring for large scale cloud systems," in *IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom 2013)*, vol. 2, December 2013, pp. 341–344.

[17] Y. Tan, V. Venkatesh, and X. Gu, "Resilient self-compressive monitoring for large-scale hosting infrastructures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 3, pp. 576–586, March 2013.

[18] J. Carpentier, J.-P. Gelas, L. Lefevre, M. Morel, O. Mornard, and J.-P. Laisne, "CompatibleOne: Designing an energy efficient open source cloud broker," in *2nd International Conference on Cloud and Green Computing (CGC 2012)*. Washington, USA: IEEE Computer Society, 2012, pp. 199–205.

[19] P. Hintjens, *ZeroMQ: Messaging for Many Applications*. Sebastopol, USA: O'Reilly Media, March 2013.

[20] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Computing Surveys*, vol. 35, no. 2, pp. 114–131, June 2003.

[21] F. Fabret, H. A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha, "Filtering algorithms and implementation for very fast publish/subscribe systems," *SIGMOD Rec.*, vol. 30, no. 2, pp. 115–126, May 2001.

[22] R. Bolze, F. Cappello, E. Caron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lantéri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E.-G. Talbi, and T. Iréa, "Grid'5000: a large scale and highly reconfigurable experimental Grid testbed," *International Journal of High Performance Computing Applications*, vol. 20, no. 4, pp. 481–494, November 2006.

[23] "Storm: Distributed and fault-tolerant realtime computation," Project Website, 2014. [Online]. Available: https://storm.incubator.apache.org/

[24] "S4: Distributed stream computing platform," Project Website, 2014. [Online]. Available: http://incubator.apache.org/s4/