Energy monitoring and billing in OpenStack based Cloud

François Rossigneux
INRIA, Lyon, France
Email: francois.rossigneux@inria.fr

Jean-Patrick Gelas
University of Lyon, France
Email: jean-patrick.gelas@univ-lyon1.fr

Laurent Lefèvre
LIP Laboratory, ENS Lyon, France
Email: laurent.lefevre@ens-lyon.fr

***Abstract*—Large data center composing the Cloud infrastructures are known for consuming a tremendous amount of energy. Operators of those infrastructure require tools for monitoring not only processing load, storage usage and network capacity, but also power drained by each node. This very last information is meaningful to better understand ressources usage, improve global efficiency of the whole infrastructure, and eventually billing customers on a "pay-as-you-use" model. To achieve this, a complete software framework and hardware architecture must be provided. In this article, we introduce our energy monitoring software framework called Kwapi. It supports several wattmeter devices, formats the measurements to ease unification, and sends them to several components for processing them. Kwapi architecture is scalable and extensible, and interacts with the famous OpenStack Ceilometer component.**

I. Introduction

Clouds are now becoming a defacto standard for supporting large scale secured remote operation (computing, storage, network). Due to their ability in aggregating several virtualized machines on physical infrastructures, clouds have the potential to help reducing the over provisioning of resources and thus could help in reducing energy consumption of these infrastructures.

Motivations:

Without monitoring and measurement tools, no system analysis can be done seriously. And without analysis, no optimization and thus energy efficiency improvements can be done.

An infrastructure which is monitored can take efficient decisions, like scheduling a resource reservation ; placing or moving virtual machines ; switching off unused resources (and power on them only when required) ; supporting power capping ; and set the air conditioner power optimally. It might also be used to do applications energy profiling to improve them on a new criteria (Joules or kWh).

Difficulties:

In brief, power monitoring open a large set of optimization perspectives. However monitoring power is not easy and straightforward. First, computing nodes does not consume energy identically even though they are of the same model, doing the same job or nothing (idle state). It means that each node must be measured individually. Second, there is currently on the market different power probes with different behaviour, connections and communications protocols. A unification work must be done. Thus, one of our challenge was to design and implement a framework which can be deployed on a large scale infrastructure (like several large data centres geographically disseminated).

This paper addresses the challenges of measuring the energy consumption of Openstack based clouds and to inject measurements in Openstack framework components like Ceilometer.

Thus, we propose to introduce our energy monitoring software framework called Kwapi. It supports several wattmeter devices, formats the measurements to ease unification, and sends them to several components for processing them. Kwapi architecture is scalable and extensible, and interacts with the famous OpenStack Ceilometer component.

This paper is scheduled as follows: Section 2 proposes a short state of the art about Energy and Clouds. Section 3 introduces briefly the hardware environment in the context of measuring servers power consumption and highlights the heterogeneity aspects we have to tackle with this type of devices. Section 4 provides a detailed study of the software side in this research work. After a brief introduction to Ceilometer, we describe the architecture and functioning of Kwapi. Then we present two plugins among others we designed. The first one interacts with Ceilometer, while the second one displays measurements in quasi real time within a web page. To close this section, we show how we take advantages of the ZeroMQ bus. Section 5 proposes a performance evaluation followed by a discussion, in section 6, about some improvements we may add in the current Kwapi implementation. Finally, section 7 concludes this paper.

II. State of the Art about Energy and Clouds

Energy and Clouds or "Green Clouds" terms refers to activities in the area of cloud computing whose goal is to design and exploit IT resources in the most efficient ways in terms of energy consumption. The goal is often twofold: first, obtaining the very best output in terms of processing, storage and communications for each consumed kWh, and second, reducing the environmental impact of the creation, operation and dismantlement of the infrastructure by taking into account the whole lifecycle of all the infrastructure elements. Basically, all the propositions made in the Green IT context fit Green Clouds requirements.

To build and run an efficient green data center, several aspects must be taken into account, ranging from hardware, to

geographic location of the facility, as well as software used to help optimizing operations.

Hardware
: While the word "cloud" is used to describe such infrastructures, it is not "vaporware", but they rely on real interconnected computers consuming a lot of resources to be built, used and eventually destroyed. The good news is that manufacturers made major efforts over the past decade to build computer resources consuming less energy while providing always more performance. They also make progress in using recyclable materials or easing the recyclability of used materials.

Resources Consolidation
: Thanks to the virtualization techniques we are able to run independently several systems on the same processing node which allows for optimal use of an infrastructure by migrating and aggregating virtual systems not heavily used to a given physical node. Virtualization considered as a common technique nowadays still represent a challenge to optimize VM placement with respect to several factors.

Monitoring
: To improve power efficiency while maintaining a good quality of services it is mandatory to monitor the infrastructure in terms of processing and storage requirement and also power drain. Without all these information, as mentioned above, it is not conceivable to improve the operation of the infrastructure.

Scheduling
: A cloud infrastructure whatever its flavor (IaaS, SaaS,...) provides services. Services are required by the end-users at a given time, immediately (or not) for a defined period of time. Then, scheduling may help in using the infrastructure efficiently.

Cooling
: modern electronic equipment can run at higher temperature... Free cooling is now an option to be considered.

Strategic geographic location and Energy sources
: A data center must be located in specific region where renewable energy (hydroelectric, geothermal, solar, wind or tidal power) is available... Networks communication should not be a problem anymore.

Networking
: Data network plays a crucial role in such infrastructures in terms of latency and throughput. But the interconnect requires also power which is correlated with performance.

Here are some initiatives worth to be mentioned (wikipedia cut'n paste mostly):

- **Green Grid** is a non-profit, open industry consortium of end-users, policy-makers, technology providers, facility architects, and utility companies collaborating to improve the resource efficiency of data centers and business computing ecosystems. With more than 175 member companies around the world, The Green Grid seeks to unite global industry efforts, create a common set of metrics, and develop technical resources and educational tools to further its goals.
- **Green500** list ranks computers from the TOP500 list of supercomputers in terms of energy efficiency. Typically measured as LINPACK FLOPS per watt.
- **GreenQloud** (Iceland) is a cloud computing services company with headquarters in Iceland, offering truly green cloud computing services powered by 100% renewable energy resources. GreenQloud's services include carbon neutral cloud server hosting, online storage, backup and cloud based computing and High performance computing services to companies, National Research and Education Networks (NRENS) and consumers.
- **Green Comm Challenge** is an organization founded and led by Francesco De Leo that actively promotes the development of energy conservation technology and practices in the field of Information and Communications Technology (ICT).

In conclusion, to build and run a real Green Cloud infrastructure all the Green IT propositions must be take into account to maximize energy efficiency and minimize impact on the environment. Alternative solutions like carbon offset credits must be considered as a green marketing measure only. Given the broad success of the cloud computing, adopted and used equally by companies and private customers, infrastructures hosting such services must definitively be efficient with minimal impact on the environment.

Finally, to the best of our knowledge, this paper is the first one to present energy consideration in clouds based on the OpenStack architecture.

III. Hardware environment

Data centers are clusters of nodes, exploited by a scheduler which coordinates the distribution of tasks among the nodes and sites. The scheduler selects the hosts by taking into account the user requirements, but also multiple criteria, including energy.

To measure nodes power consumption, several theoretical power models have been proposed. However, real measurements shows that there is always significative differences between theoretical values and real values measured empirically. For example, we have observed a difference up to 20% in energy consumption on four identical nodes (Dell R610), bought the same day, located in the same room, and running OpenStack Folsom on GNU/Linux Ubuntu 12.04. The reasons for that are not obvious. This is why we focus our attention on real hardware to do measurements and show their advantages and limitations.

Available wattmeters on the market are very heterogeneous in terms of links, communication protocols, packaging and quality of measures. They are mostly packaged in multiple outlet power strip called PDU or ePDU (see Figure 1, right), or more recently in the IPMI cards embedded in computers,

Fig. 1. (Left) Prototype of wattmeter made by OmegaWatt for monitoring 150 nodes in our data center. (Right) Outlet power strip (ePDU) which allow to query each outlet individually.

TABLE I
DEVICES CHARACTERISTICS DIFFER GREATLY

Name	Protocol(s) / link(s)	Frequency (s)	Resolution (W)
Eaton	Serial or SNMP / Ethernet	5	1
Schleifenbauer	SNMP / Ethernet	3	0.1
OmegaWatt	"IrDA" / Serial	1	0.125
Dell iDrac6	IPMI / Ethernet	5	7
Watts Up?	Legacy / USB	1	0.1
ZES LMG450	Serial	0.05	0.01

used initially as an alternative to shutdown or power up the computer chassis remotely. The type of links in use are either Ethernet to transport IPMI or SNMP packets over IP, or USB or RS-232 serial links. Wattmeters relying on Ethernet are generally linked to the administration network (off the data center customer's network). Moreover, wattmeters may have various behaviours. Some of them send measurements on a regularly time basis (push), while some others must be queried (pull). Among characteristics we may name are: the maximum number of measurements per seconds (with a lot of measurements per second, you will get a precise idea of your application's energy profile), resolution of the value, sensibility of the probes and finally the methodology applied for each measure (e.g. mean value between several measures, instantaneous value, exponential moving average value). Table I shows some characteristics of the devices we had the chance to evaluate on our data center nodes.

IV. SOFTWARE ARCHITECTURE

Wattmeters generates a large amount of data that need to be stored somewhere. The first approach would consist to store the data on the monitored compute nodes, while the second one would be to centralise all metrics in one place. Local storage is more robust and scalable, but it has two major drawbacks: the latency when the data must be retrieved and the risk that the data be unavailable (if the machines are in sleeping modes). Centralised storage allows to access and process the data quickly, but could generate more network traffic given that all measurements need to be sent continuously over the network. But once they are centralised, the network traffic is minimised whatever the number of query to these measurements (through a specialised component like Ceilometer).

We have chosen the centralised storage approach, while minimising the network traffic and preserving the scalability. Ceilometer is used to store our power consumption metrics, and we propose an architecture to retrieve values from wattmeters, and send them to Ceilometer. In the following section we will describe Ceilometer followed by Kwapi.

A. OpenStack Ceilometer

Ceilometer is the OpenStack's framework for collecting metrics, also used for billing. It has two types of agents: the compute agent and the central agent.

The compute agents are running on each compute node. They retrieve the resources usage related to a given instance and a given resource owner, while the central agent executes pollsters on the management server to retrieve the data that are not related to a particular instance of calculation.

All metrics are published on the internal bus of Ceilometer as counters (cumulative type, gauge or delta). This bus is listened by several modules, like the Ceilometer Collector which then stores these counters in a database. This database is searchable via Ceilometer API, and allows to view the history of a resource's metrics.

In the context of energy metrics publication, we use the central agent and a dedicated pollster we developed. It queries the Kwapi API plugin and publishes cumulative (kWh) and gauge (W) counters. These counters are not yet associated with a particular user, since a server can host multiple clients simultaneously.

B. Kwapi

Kwapi is our framework, designed for acquiring energy consumption metrics. It allows, among other, to upload metrics from the wattmeters to Ceilometer.

Its architecture is based on a layer of drivers, responsible for the acquisition metrics, and a layer of plugins that collect these metrics. The communication between these two layers goes through a bus. In the case of a distributed architecture, a plugin can listen to several drivers at remote locations (see Figure 2).

Drivers and plugins are easily extensible to support other types of wattmeters, and provide other services.

1) Drivers layer: The drivers are threads started by a manager, which instantiate them with a set of parameters loaded from a configuration file (unified with the OpenStack configuration file format, similar to INI). These parameters are used to query the meters (IP address, port, etc.) and indicate the sensor IDs in the issued metrics. The metrics are Python dictionary with a set of fields. Optional fields can be added, such as voltage, amperage, etc.. The metrics are signed.

The manager periodically checks if all threads are active, and restart them if necessary (incidents may occur, for example if a meter is disconnected or becomes inaccessible). The drivers can manage incidents themselves, but if they finish

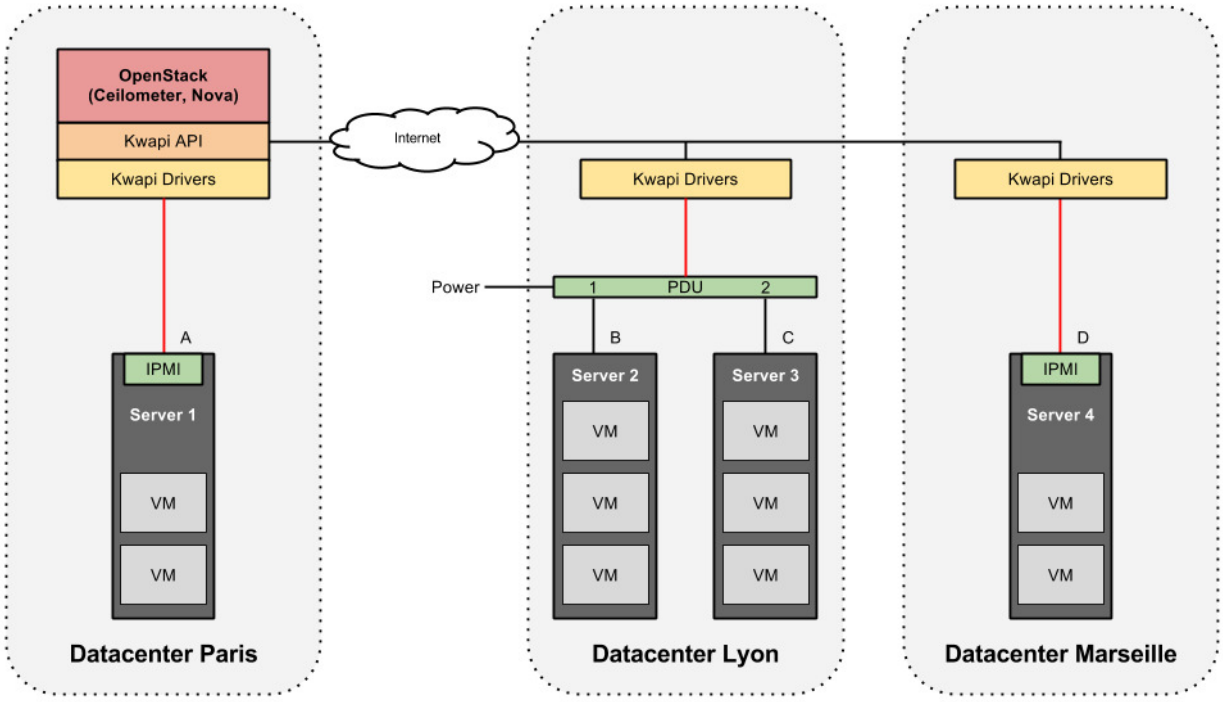

Fig. 2. Software components in the global architecture

their execution, it does not matter because they will be automatically restarted by the manager. It is important to avoid losing measurements because the information reported is watts and not kWh: if a value in watts is lost, we lose information.

2) Plugins layer: The plugins retrieve and process the metrics sent by the drivers on the bus. They expose them to other services (Ceilometer) or user (visualization). They can subscribe to all sensors, or just some of them, through a system of prefixes. After verifying the message signature, they extract the fields, and process the received data. Currently Kwapi has two plugins: a plugin API for Ceilometer, and a visualization plugin.

a) API plugin for Ceilometer: The API plugin computes the number of kWh of each probe, appends a timestamp, and stores the last value in watts. These data are not stored in a database, as Ceilometer already has one. If a probe has not issued metrics for a long time, the corresponding data are removed. This plugin has a REST API that allows to retrieve the name of the probes, and W, kWh and timestamp. This API is secured with OpenStack Keystone tokens: the client provides a token, and the plugin contacts Keystone API to check the token validity before sending its response.

b) Visualization plugins: The visualization plugin (Figure 3) builds RRD files from received metrics, and generates graphs. Each graph shows a plot of the energy consumption in a given period, with additional information (average electricity consumption, minimum and maximum watts, last value, total energy and cost in euros). A web interface displays the generated graphics. A cache mechanism triggers the regeneration of graphics only if they are outdated, during the consultation. RRD (Round-Robin Database) files are of fixed size, and store several collections of metrics with different granularities.

3) Internal bus: ZeroMQ: Kwapi uses ZeroMQ, a fast brokerless messaging framework (transmitters play the role of buffers), written in C++. It supports a wide range of bus: cross-thread communication, IPC, and TCP. Switching from one to another is very simple. ZeroMQ provides several design pattern (publisher/subscriber, request/response, etc). In our architecture, we use a publisher/subscriber design pattern. Drivers are the publishers, and plugins are the subscribers. Between them, one or more forwarders simply forwards the packets, and broadcasts a packet to all the plugins which have subscribed to a given probe. Thanks to the forwarders, the network usage is very optimised because the packets are sent only once, regardless the number of plugins that listen a probe. If a probe is not listened by any plugin, its measurements are neither sent over the network nor to the first forwarder. The forwarders not only reduce dramatically the network usage, but allow to build flexible architectures, by bypassing networks isolation problems, or doing load balancing. For example, in the Figure 4, we could imagine that the link between Karlsruhe and Stuttgart is less congested than the one between Karlsruhe and Lyon. So rather than to establish a connection between Karlsruhe and Lyon (and thereby duplicate the packets sent between France and Germany), it is preferable to rely on the Stuttgart node (supposed to be always powered on). In this configuration, the Karlsruhe node needs to know only the address of the Stuttgart node. But this very last one needs

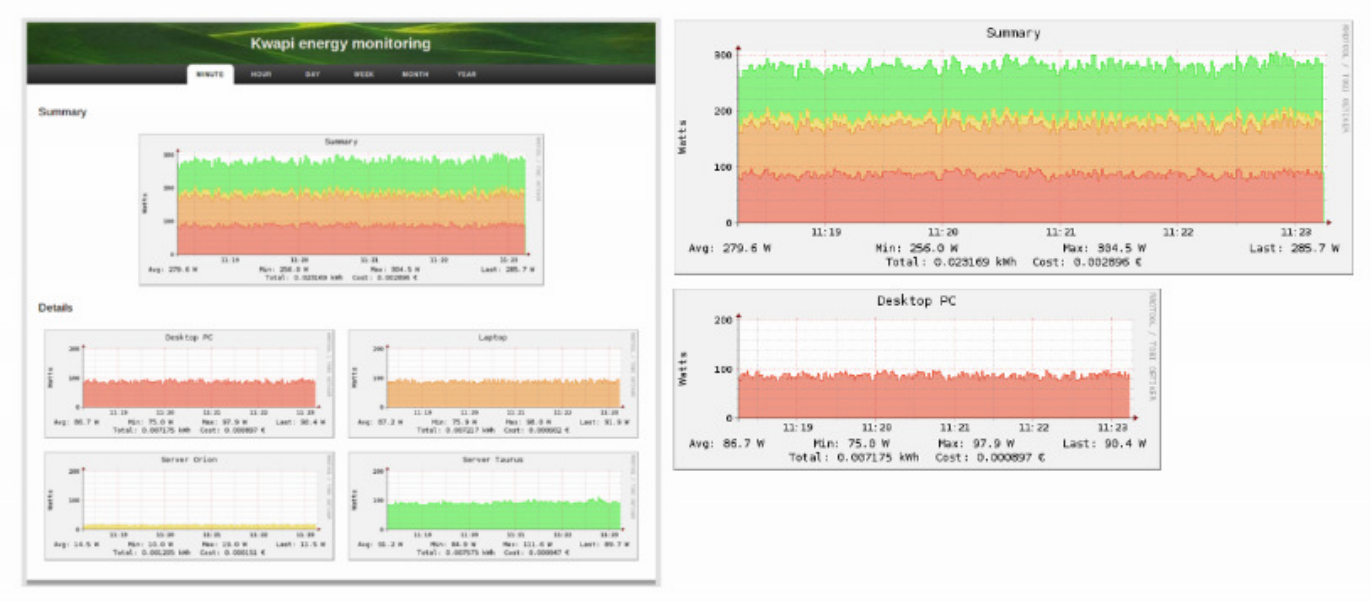

Fig. 3. Snapshot of a webpage displaying the visualisations plugin in action

to know the addresses of the two french nodes.

In a very large architecture with thousands of nodes, it is recommended to run severals servers with drivers, and severals servers with plugins. Ceilometer is then able to collect the metrics on severals Kwapi API plugins.

V. Performances evaluation

We can evaluate Kwapi in terms of CPU, memory and network bandwidth usage.

A. CPU and memory

The CPU and memory usage depends very much on the drivers, plugins amount and complexity, and on the signing of messages (enabled or not). If it lacks compute power or memory, it is easy to add more servers.

B. Network

In a simple architecture where the drivers and the plugins layers are on the same machine, the network traffic is minimal. In a distributed architecture with low bandwidth, using many forwarders decrease the network usage (no redundant packets sent) but weakens the system (in case of forwarder node failure).

The plugins can select the probes they want to watch, so any useless traffic is eliminated.

To decrease the header overhead, it is better to send large packets. The drivers, on their side, can build lists of measures, and send them to the forwarder at one stroke. For example, this is used in the PDU drivers with multiple outlets. And ZeroMQ, on its side, has its own optimization mechanism: if several drivers send metrics simultaneously, ZeroMQ aggregates them in one packet. In our experiments, some packets contain up to ten metrics. The metrics are Json dictionaries, which has the advantage of being human readable and easily parsable, while keeping a very small surcharge. The size of those dictionaries may vary, depending on the number of fields set by the drivers (signing add some overhead), while the ACKs have a fixed size of 66 bytes (on a TCP link).

VI. Limitations

We can distinguish two types of limitations: the ones that are inherent to the current implementation of Kwapi, and those which are high level limitations that are not easily bypassable without future innovations and new paradigms.

A. Kwapi limitations

1) Consumption accuracy if some messages are lost: Losing a message between the driver and the plugin layers diminish the power consumption accuracy: the plugin will generally suppose that the consumption is identical to the last message properly received. To overcome this limit, we could compute the kWh in the driver layer. Or we could add some redundancy in packets. But that induce additional costs that would need to be compared with the packet loss rate.

2) Automatic acquisition of IPMI cards IP addresses: The automatic configuration of the Kwapi drivers layer is a challenge: there is no trivial mechanism for making correspondence between a probe and the host ID. In addition, the parameters to query the wattmeters are now defined manually in the Kwapi configuration file. In the future, we intend to implement a CMDB (configuration management database), which would retrieve all properties attached to a host ID, including the wattmeter configuration parameters (IPMI IP address, outlet identifier in the case of a PDU, etc). However, this database will need to be fed, probably by hand (at least regarding the wattmeters).

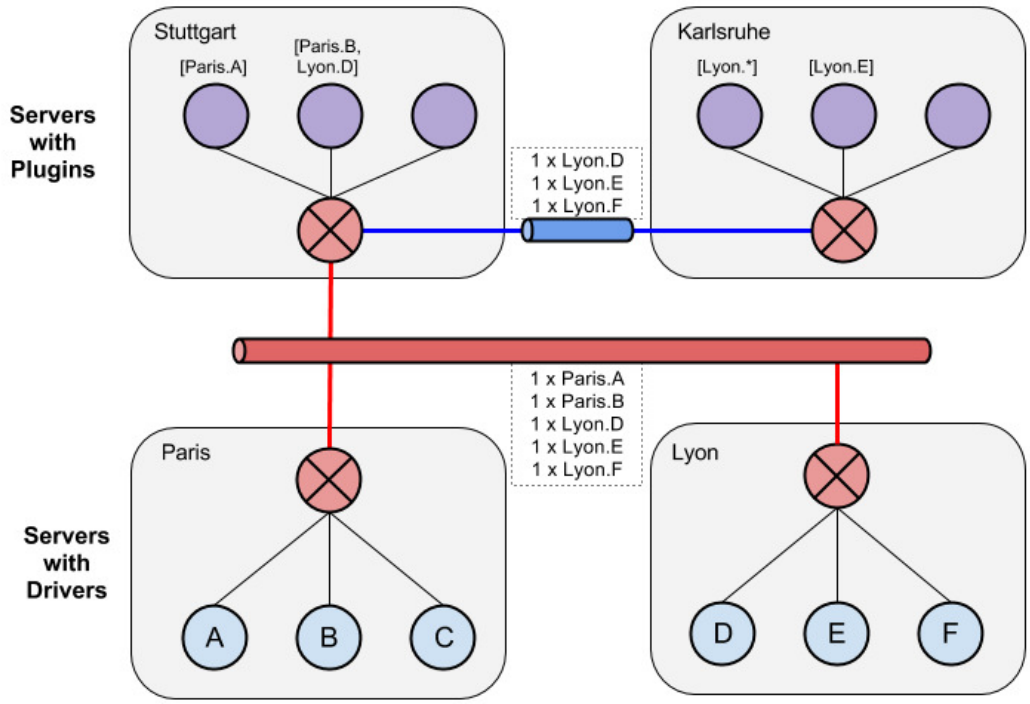

Fig. 4. The tree architecture of ZeroMQ

3) Max open file descriptors limit: Each driver is a thread because the creation of hundreds of process would be unnecessarily burdensome. In addition, it allows the use of cross-thread communication in ZeroMQ, which increases efficiency (no IPC socket).

The number of open files per process being limited, we may encounter this limit if a large number of drivers are loaded. It is possible to overcome the limit by setting a parameter in /etc/security/limits.conf, to adjust the hard limit (for the whole system) and soft limit (per process) accordingly. At last, ZeroMQ then has its own limit, which is set when it is instantiated. But this limitation is rarely a problem, because the server will be probably overloaded before reaching this limit (in a such situation, simply spread the drivers among several servers).

4) Viewing the energy consumption of thousand compute nodes: The current visualization plugin that we designed and introduced above is useful for small-scale infrastructures, but it would be inappropriate for a huge data center. Moreover, we should consider how to graphically represent the consumption of several thousands of nodes.

B. Other limitations (or ongoing works discussion)

1) Power consumption of a virtual machine (VM): Attributing energy consumption to a client is easier if there is only one client per machine. But in a multi-tenancy environment, this is much more difficult. We can rely on models to estimate the consumption of VMs, but we can not prove the accuracy of the results to the client. In addition, a single client on a machine may be penalized because the static cost of the machine will not be divided between several clients. We must find a fair distribution of static costs.

2) Network energy cost (per switch, and also per stream): For billing purposes, it would be normal to charge according to the network cost incurred.

A simple approach would be to look at the volume of data exchanged by the network adapter. But that does not take into account the path of the packets in the network (number of jumps), nor the network equipment efficiency. One could imagine to append a marker of the packets: so, by receiving packets, it would be easy to know the cost. But what about the packet sent? The receiver should inform the source of the cost involved. And how to charge the network traffic? Shall we charge the source or the receiver, or both?

VII. Conclusion and Future works

Measurements stored in the Ceilometer database will be processed by several modules, which are currently under development:

- An assignment module to assign metric to the users: metrics are tight to a machine, but not the user who used it. This module will do this association, allowing the users to know their energy consumption, and the provider to charge it (easier with one user per machine).
- A ranking module: identical machines do not have exactly the same energy consumption, and their fans may run faster or slower. This module will allow to retrieve the most efficient machines in a set of identical machines. It will be used by the scheduler, and a power saving mode module.

- A scheduler that takes into account the energy efficiency machines. For this, the consumption of the machines will be compared with their performance (flops / W).
- A power saving mode module: in a set of identical machines, the less efficient ones will be shut down.

ACKNOWLEDGMENT

This research is supported by the French FSN (Fonds national pour la Société Numérique) 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://www.grid5000.fr). Authors wish to thank Julien Danjou for his help during the integration of Kwapi with Openstack and Ceilometer.

REFERENCES

Not yet available.