Document	FUI-10-COMPATIBLE ONE		
	Livrable L13.3		
Date prévue	01/06/2012	**Nature**	Interne
Date livraison	12/07/2012		
Statut	Final	**Version**	1.0

Propriétés du Document

Source du Document	FUI-10-COMPATIBLE ONE
Titre du Document	Energy management system and energy consumption efficiency - COEES Code v1
Module(s)	COEES
Responsable	Laurent Lefèvre
Auteur(s) / contributeur(s)	Julien Carpentier & Maxime Morel
Statut du Document	Final
Version	1.0
Validé par	
Date de la validation	11/07/2012

Résumé

Following document highlight different research and implementation for COEES module and a first version of energy efficiency module code.

Mots Clefs

Energy efficiency, Cloud Computing, green IT, CompatibleOne

Summary

1. Introduction

Adapted energy monitoring to virtualized infrastructures is a new and complex subject. Our work is divided between a constant monitoring of technological development around green IT, cloud computing and implementation of the CompatibleOne energy modules. We develop a system to collect energy consumption information that enables access to information on the energy consumed by physical elements used in the cloud, as well as the evaluation of the energy bill, and the environmental impact.

In CompatibleOne context, energy monitoring provides an additional SLA criterion for Broker choice in virtual machine placement, a precise consumption billing and energy efficiency for Cloud Service Provider with live energy monitoring of their platforms.

An abstraction of energy collect is available through HTTP REST primitive. According to energy usage, some modules need an important precision and others just an average consumption for energy usage prediction or define Cloud Service Provider green “reputation” score.

2. Probes specifications

For this informations gathering, we preconize physical probes because just an estimation is not enough and the system needs to get precise energy every 3-4 seconds. Moreover, cloud provider needs an easy integration for their existent cluster. We studied different industrial probes with precision and measures frequency testbed.

Eaton ePDU		Voltage : 230 V Current : 16 A Outlets : C13, C19 Interface : serial, ethernet (SNMP) Measure frequency : 1 value every 5 seconds Measure precision : 1 W
Schleifenbauer ePDU		Voltage : 230 V Current : 16 A Outlets : C13, C19 Interface : ethernet (SNMP, Modbus, MySQL) Measure frequency : 1 value every 3 seconds Measure precision : below 0.1 W
Dell iDRAC6 : IPMI		Measures : internal sensors (Dell proprietary sensors) Interface : IPMI Measure frequency : 1 value every 5 seconds Measure precision : 7 W
Wattmetre (OmegaWatt)		Measure frequency :1 value per minute Measure precision : below 1 W Interface : serial

Product	Interface	Pros	Cons
OmegaWatt	Serial	known probes (GRID'5000 platform)	Not scalable
Eaton	SNMP v1	Outlet measure, integration, precision	-
Schleifenbauer	SNMP v1	Outlet measure, integration, precision	-
IPMI Addon card	IPMI Interface	Generic in most one rack server	precision

Eaton and Schleifenbauer ePDU[2] seem to be the best choice because we get all informations we need like instant consumption, power factor, efficiency through SNMP[3] protocol. IPMI[4] will be included into our solution because it is currently the most deployed solution in cloud providers clusters for monitoring including energy.

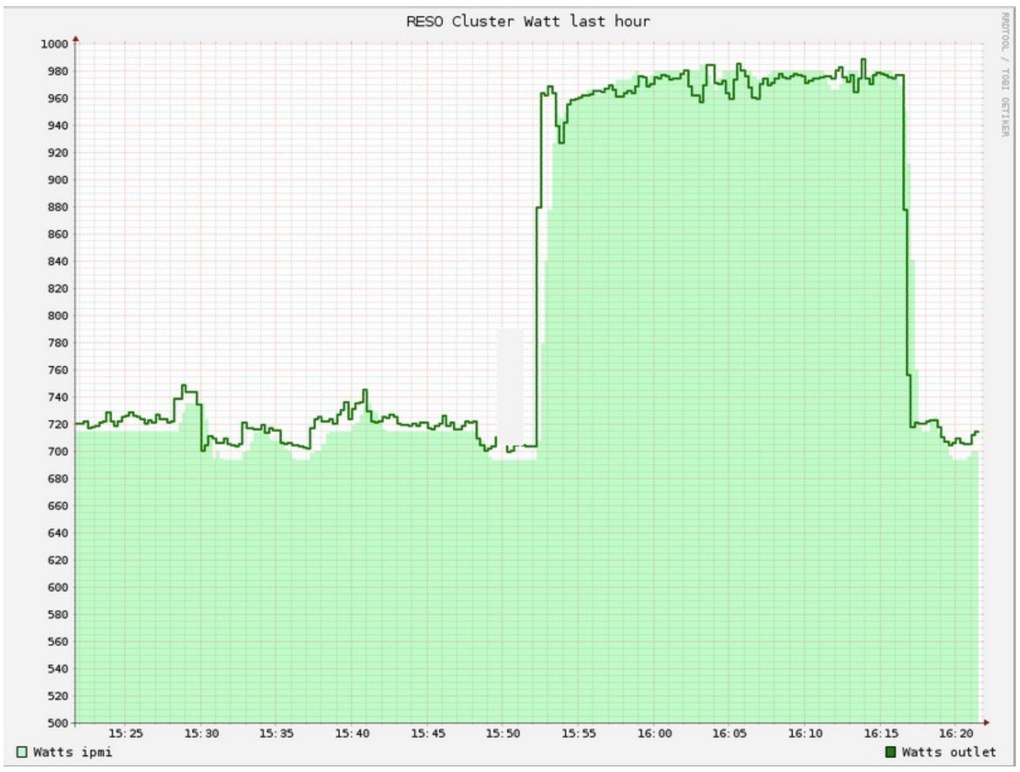


Illustration 1: Precision benchmark with IPMI and Outlets measures

However, values with IPMI is below outlet measure as shown on this energy consumption graphic where dark green curve represent outlet measure and light green zone represent IPMI (figure above).

3. Monitoring System & Implementation choice

Monitoring system is time-based and large scale so it's crucial to use common time to link events with timestamp so we choose POSIX time (ISO8601).

3.1. General Architectures

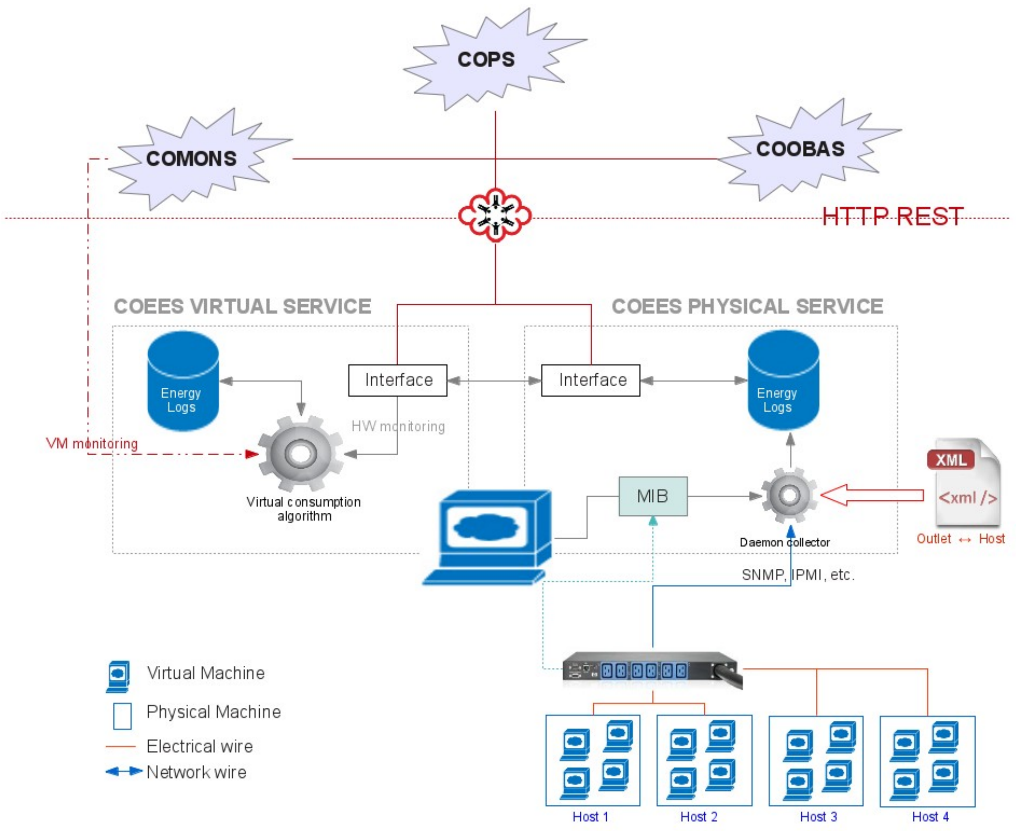


Illustration 2: Energy Monitoring Modules Architecture v2

On this diagram we underscore the necessity to split hardware monitoring and virtual machine monitoring provisioned on-demand through cloud provider IaaS[5] system like OpenStack or OpenNebula.

Indeed, physical host consumption comes directly from probes through daemon collector request because we only need mapping between physical host and outlet and configuration is pretty stable. Virtual host life is based on user needs and we need to keep a trace of virtual host user, physical host consumption and context (CPU usage, IO, etc.) to estimate virtual host consumption.

3.2. Software Architectures

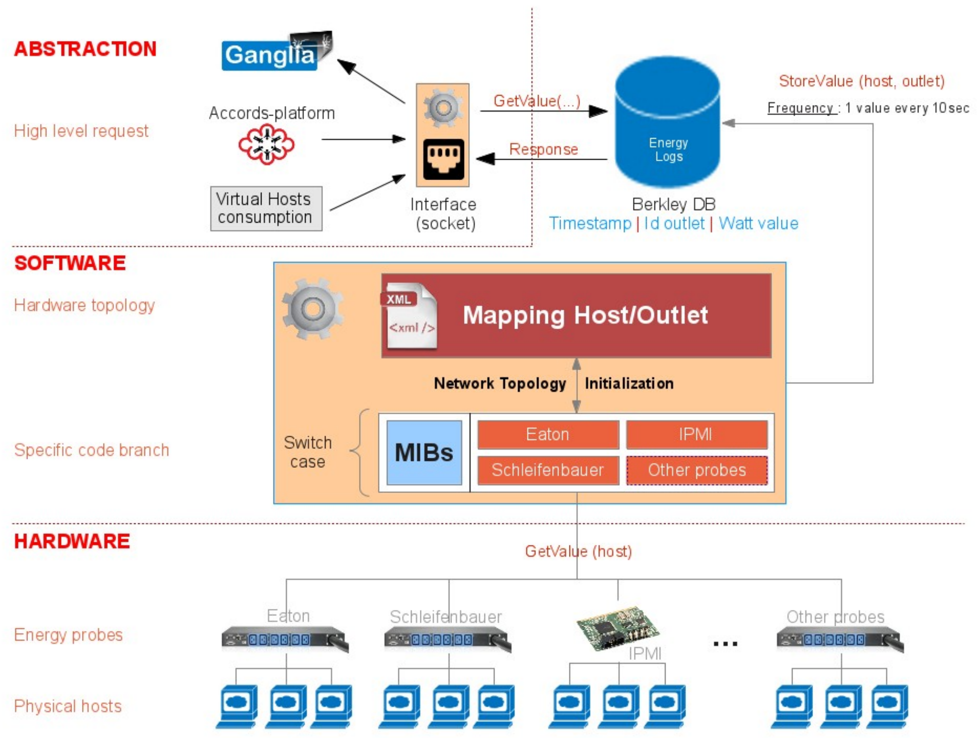


Illustration 3: Software Architecture COEES v3

Energy monitoring system in CompatibleOne is designed on three different layers. The purpose of these layers is to have modularity and abstraction. We will review these layers.

First layer is the physical machines monitored. Just above we have the energy monitoring probes, which can be different. Typically, there is PDU or IPMI addon cards which come from different vendors. Energy monitoring probes are most often heterogeneous.

So in the next layer, we have a collecting daemon which can handles all these probes. Energy consumption values are gathered from the probes below. This daemon must run on a physical machine reachable by cloud provider in order to reach probes through network (local network or VPN[6]).

Provider administrators must fill a XML[7] file describing energy monitoring topology : mapping between machines (hostname or ip) and outlets. For each machine, this mapping gives us the information on which probe it is plugged and all information needed to communicate with this probe. Each type of probes could own different access protocols. For example, most of PDU use SNMP protocol. Each vendor use its specific MIB[8], so we need to know the OID[9] name to have required value, like watt.

Nota bene : some PDU do not implement directly watt value and we have to compute it using voltage, current and power factor value.

Formula used in SNMP request for watt consumption :
*Watt = U (volt) * I (ampere) * Pf (power factor)*

Schleifenbauer example :

```
U = snmpget -v 1 -c public 192.168.0.203 actualVoltage0.6.1
SPGW-MIB::actualVoltage0.6.1 = INTEGER: 232.8
I = snmpget -v 1 -c public 192.168.0.203 actualCurrent0.6.1
SPGW-MIB::actualCurrent0.6.1 = INTEGER: 0.4
Pf = snmpget -v 1 -c public 192.168.0.203 powerFactor0.6.1
SPGW-MIB::powerFactor0.6.1 = INTEGER: 86.9 %
```

Watt value = 80.82

Eaton example :

```
U = snmpget -v 1 -c public 192.168.0.202 outletVoltage.19
SNMPv2-SMI::outletVoltage.19 = INTEGER: 481
I = snmpget -v 1 -c public 192.168.0.202 outletCurrent.19
SNMPv2-SMI::outletCurrent.19 = INTEGER: 0.48
Pf = snmpget -v 1 -c public 192.168.0.202 outletPowerFactor.19
SNMPv2-SMI::outletPowerFactor.19 = INTEGER: 75 %
```

Watt value = 173.16

So, using all these mapping information, we can use our generic function GetValue (host) which will handle all the specificities of the probes below. Next, we store the data retrieved from the probes using a function StoreValue (host, outlet) in a Berkeley DB[10] using this structure : Timestamp | id_outlet | watt value

Outlet/Hosts mapping XML example :

```
<cluster name="RESO">
   <probe model="schleifenbauer" ip="192.168.0.203">
          <mib filename="33_0_FR_SPGW_MIB.mib">
                    <field name="voltage"        node="actualVoltageO"     type="integer"/>
                    <field name="current"        node="actualCurrentO"     type="integer"/>
                    <field name="powerfactor"   node="powerFactorO"            type="integer"/>
          </mib>
 <hosts>
                    <host ip="10.5.5.8" outlet="1"/>
                    <host ip="10.5.5.9" outlet="2"/>
 </hosts>
   </probe>
   <probe model="eaton" ip="192.168.0.202">
 <mib filename="EATON.mib">
          <field name="voltage"             node="outletVoltage"              type="integer"/>
          <field name="current"             node="outletCurrent"              type="integer"/>
          <field name="powerfactor"    node="outletPowerFactor"     type="integer"/>
 </mib>
 <hosts>
                    <host ip="10.5.5.10" outlet="19"/>
                    <host ip="10.5.5.11" outlet="20"/>
 </hosts>
</probe>
</cluster>
```

Because a Berkeley DB is a simple file it can be reopened by the next layer of the system which handles the high level requests. This is done using another software daemon. The goals of this layer are to answer high level request or to export data to other services like exposition to ganglia or virtual machine consumption service. High level request are coming from the other modules of CompatibleOne. For example elasticity or billing services which will ask about energy consumption. The communication is made using an OCCI[11] interface.

3.3. Interface with Ganglia Monitoring System

a) Presentation & Usage

Ganglia, open-source project that grew out of the University of California, is a scalable distributed monitoring system for high-performance computing systems such as clusters and Grids. It is based on a hierarchical design targeted at federations of clusters. It leverages widely used technologies such as XML for data representation, XDR[11] for compact, portable data transport, and RRDtool[12] for data storage and visualization.

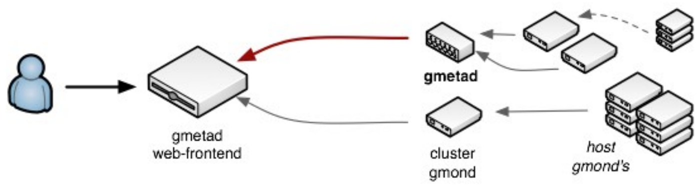
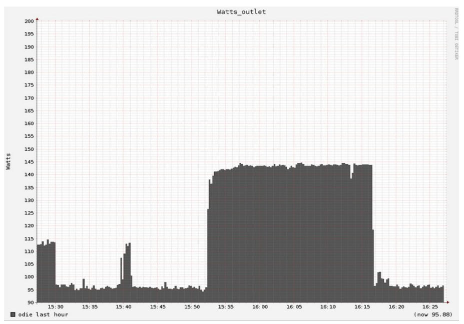


Illustration 4: Ganglia Monitoring System architecture

In our context, we use it as management and verification tool for energy data stored in database concerning virtualized host in cloud environment . This tool ensure possibility for an administrator to display aggregated values per days, weeks or months on single-node or a whole cluster.



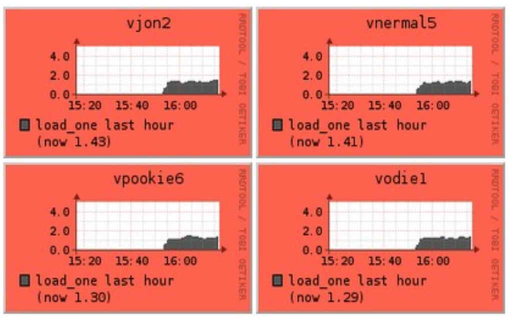


Illustration 5: Single-node watt consumption with Ganglia

Illustration 6: Multi-node watt consumption with Ganglia

b) Technical operation

The Ganglia Metric Client (gmetric) announces a metric value to all Ganglia Monitoring Daemons (gmonds) that are listening on the cluster multicast channel. Gmetric has the ability to spoof metrics for a host that is not running gmond. This is particularly useful in our case because we need to collect only one value on different probes on the same host who run CompatibleOne Energy Efficiency Services. Thanks to this process, energy data is stored in ganglia as if all hosts give themselves energy values.

```
gmetric --name Watts_outlet --value WattsValue --type int32 --unit
Watts -S "ip:host"
```

Last part consists of some frontend modifications in order to integrate new metric and associated graphs in PHP[13] script and HTML[14] pages.

c) Benchmark example

Ganglia allowed us to observe virtual host consolidation experimentation with a simple case where five virtual hosts run on a single physical host (blue round) and on the other hand five virtual hosts run on five different physical hosts. As we can see, with consolidation of virtual hosts on the same physical host we improved energy efficiency.

Nota bene : in our case, all physical hosts stay switched on, so idle energy consumption aren't saved.

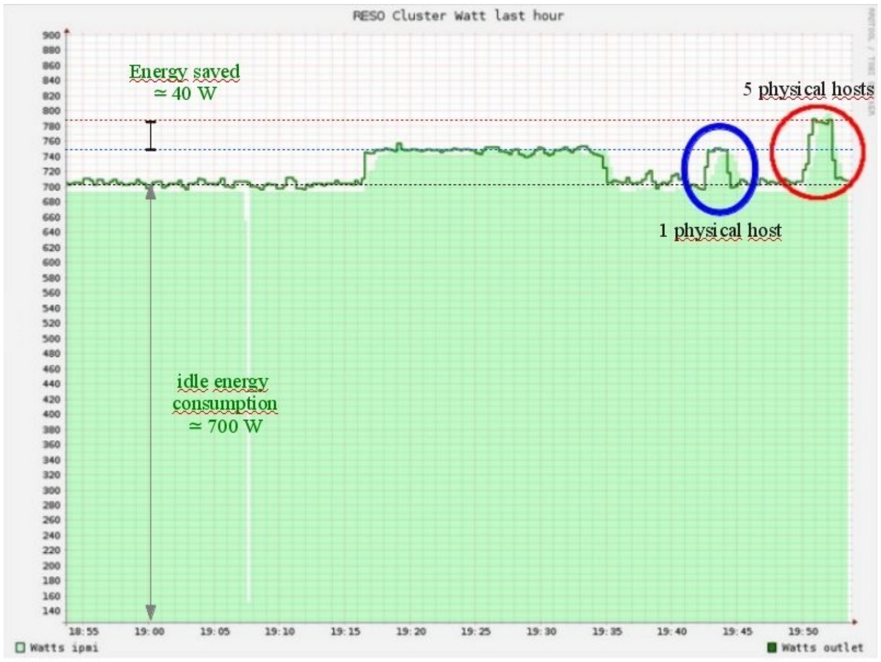


Illustration 7: Ganglia virtual host energy consolidation experiment

3.4. Glossary

1 Service Level Agreement
2 Simple Network Management Protocol
3 Power Distribution Unit with user interface management
4 Intelligent Platform Management Interface
5 Infrastructure as a Service
6 Virtual Private Network
7 eXtensible Markup Language
8 Management Information Base
9 Object Identifier
10 Intelligent Platform Management Interface
11 Open Cloud Computing Interface
12 eXternal Data Representation
11 RDDtool : Round Robin Database tool
11 PHP : Hypertext Preprocessor
11 HyperText Markup Language