Propriétés du Document

Source du Document	FUI-10-COMPATIBLE ONE
Titre du Document	Spécifications – Intégration des composants gestion de l'efficacité énergétique - Spec
Module(s)	COEES
Responsable	Laurent Lefèvre (INRIA RESO, ENS Lyon)
Auteur(s) / contributeur(s)	Julien Carpentier, Maxime Morel, Laurent Lefèvre (INRIA RESO, ENS Lyon)
Statut du Document	Final
Version	1.0
Validé par **Date de la validation**	 jj/mm/aaaa

Résumé

Présentation du framework de collecte d'énergie COEES (Compatible One Energy Efficiency Services) et de son niveau d'abstraction utilisateur permettant l'interaction via une interface web.

Mots Clefs

Gestion d'énergie, Monitoring, Green IT, CompatibleOne, INRIA RESO

Table des Matières

1 Introduction

Le document suivant (Livrable CompatibleOne L31.1- RESO) a pour but de présenter le module COEES (Compatible One Energy Efficiency Services) développé par l'équipe INRIA RESO, ses fonctionnalités et les différentes interactions. Ce module a été développé dans le cadre du projet CompatibleOne afin de fournir les composants logiciels autorisant la mise en œuvre de solutions logicielles de *cloud* efficaces en consommation énergétique. Nous détaillerons ici l'architecture globale du module, le fonctionnement du démon de collecte assurant la remontée des données énergétiques au sein du système ainsi que nos choix d'implémentation.

2 Architecture du module

La figure 1 suivante présente le composant COEES développé par l'équipe INRIA RESO :

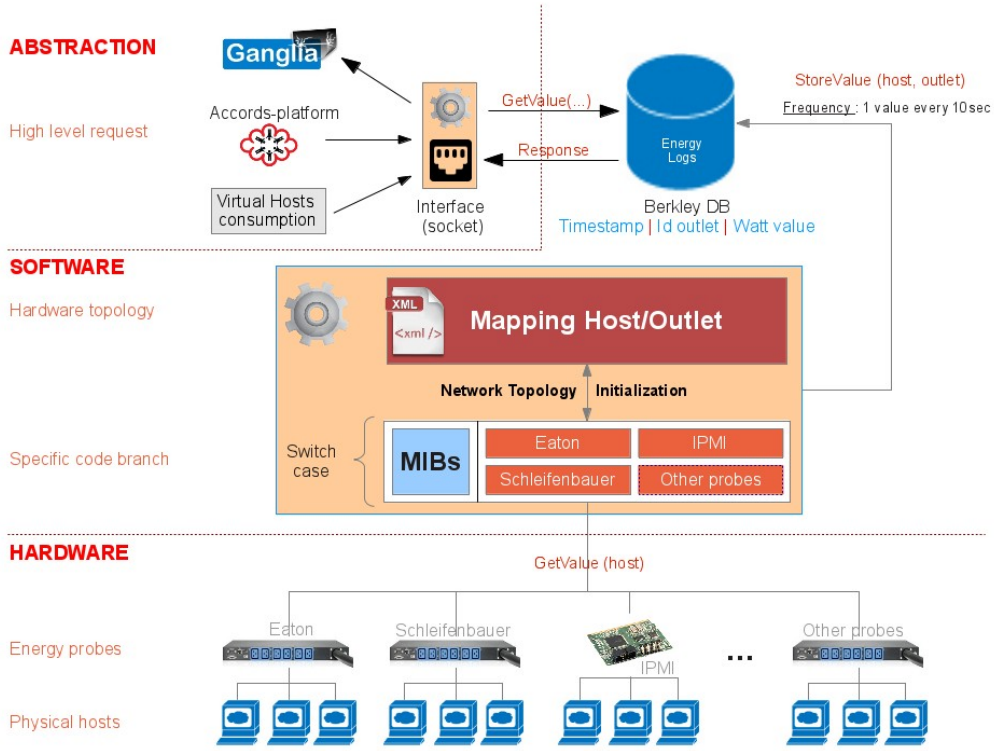

Figure 1: Architecture générale du module COEES

Le premier niveau correspond au matériel avec les machines physiques monitorées par les sondes elles-mêmes rattachées au réseau d'administration et remontant les mesures d'énergie. Un ensemble de sondes de mesures a été considéré par l'équipe INRIA RESO. En considérant les aspects facilité de déploiement ou encore de compatibilité, nous préconisons l'utilisation de sondes de type ePDU (*ethernet Power Distribution Unit*) permettant l'interrogation via le réseau en SNMP (ex : Eaton, Scheifenbauer).

Le second niveau correspond au démon logiciel de collecte, il s'agit d'un processus développé en C sous Linux assurant la récupération des valeurs de consommation électrique instantanée (watts) et leur stockage dans une base de données *BerkeleyDB*. C'est également à ce niveau que s'effectue la correspondance entre le numéro de prise du ePDU et la machine physique associée avec un fichier de configuration.

Le DCIM (*Data Center Infrastructure Manager*) connaissant la topologie et l'architecture de son *data-center* doit configurer un fichier XML d'association permettant ainsi de préciser le modèle des sondes physiques, le moyen de les interroger (adresse IP, MIB, etc).

3 Structure logicielle

3.1 Stockage

Un base de données de type *BerkeleyDB* ***(annexe 1)*** est déployée afin de permettre du stockage de type clé / valeur.

Sa structure est : id de machine | timestamp | consommation en watt

Au dessus de cette base, COEES supporte des fonctions d'accès simples :

- récupération d'une valeur de consommation pour une machine donnée selon un *timestamp*
- récupération de toutes les valeurs de consommation pour une machine donnée dans sur un intervalle de temps (le nombre de valeurs retourné dépendra de l'échantillonnage)
- récupération de la valeur de consommation moyenne pour une machine

3.2 Démon de collecte

Pour mesurer la consommation énergétique d'une machine physique, il est nécessaire de déployer des sondes matérielles externes (ex : ePDU, carte IPMI) par ressource informatique. Comme défini précédemment, ces sondes peuvent le plus souvent être interrogées par le réseau.

Dans un premier temps, il est nécessaire de mettre en place la correspondance machine physique-sonde matérielle pour savoir où faire une requête pour obtenir la consommation d'une machine parmi les autres dans un data-center. Dans le cas du *cluster* d'expérimentation de l'équipe INRIA RESO, composé de 6 nœuds physique avec OpenNebula, nous utilisons un fichier de configuration XML (map.xml) renseigné manuellement.

Pour fonctionner, le programme a besoin de connaître l'adresse IP de la machine physique sur laquelle la machine virtuelle (VM) est lancée.

Le fonctionnement est le suivant :

- La première étape du processus consiste à parcourir le fichier map.xml avec la libxml2 (*XML parsing*) pour trouver les informations sur le modèle de la sonde reliée à la machine physique. Grâce à ce procédé le processus l'adresse IP de la sonde à interroger, la MIB (M*anagement Information Database*) associée et toutes les informations nécessaires à son interrogation sont connue par le programme.

- Le démon assure ensuite le stockage de ces informations liées aux sondes dans une classe *ProbeList*, qui est un tableau de Probe. La classe Probe est dérivée pour chaque type de sonde supportée : *Probe{IPMI*, *SNMPeaton*, *SNMPschleifenbauer}*

Chaque Probe contient un tableau d'hôtes (classe *Host*). La classe *Host* contient les informations nécessaires à l'identification d'une machine, à savoir son adresse IP et le numéro de prise (*outlet*) sur le ePDU.

- Une fois le type de sonde connu, le programme utilise la bonne branche de code, qui permettra d'interroger la sonde. Du point de vue du code, il s'agit d'une redéfinition de la fonction *getValue*. La valeur de consommation de la machine physique est obtenue avec une requête sur la sonde avec les informations prises dans le fichier map.xml ***(annexe 2)***.

- A ce niveau d'exécution il est possible d'estimer la consommation de la VM en se basant sur la consommation électrique de la machine hôte. Pour cette estimation, il est possible d'inclure plusieurs métriques telles que l'utilisation CPU, l'utilisation des IO (disques, réseau, mémoire, etc.)

- une fois la consommation instantanée obtenue, elle est stockée dans la base *BerkeleyBD* pour traitement ultérieur et envoyée à *Ganglia* par *spoofing* ***(annexe 3)***.

Spécifiée de *ProbeSNMPeaton* / *ProbeSNMPschleifenbauer* :

Pour les ePDU, l'accès se fait avec le protocole SNMP. Pour cela il faut interroger le ePDU en spécifiant l'IP et le nœud. Par exemple, pour avoir l'ampérage, on interroge le nœud *outletCurrent.18* pour la machine 10.5.5.10 et *outletCurrent.19* pour la machine 10.5.5.11.

Pour calculer la consommation instantanée en watt, on effectue le calcul suivant :
P = U x I x Pf avec **P** en watt, **U** en volt, **I** en ampère et **Pf** un coefficient entre 0 et 1. **Pf** correspond au facteur de puissance.

Spécificité de *ProbeIPMI* :

Nous utilisons un fichier de cache IPMI. Ce fichier permet de réduire le temps de requête de consommation le rendant instantané contre environs 3 secondes sans utiliser de cache.

Il suffit de générer ce fichier de cache une seule fois qui est par la suite mis à jour lors des requêtes suivantes.

Génération du fichier de cache (ipmi192.168.0.80.cache) :

```
ipmitool -I lanplus -U root -H 192.168.0.80 -P pass sdr dump
ipmi192.168.0.80.cache
```

Requête pour avoir une valeur de consommation :

```
ipmitool -S ipmi192.168.0.80.cache -I lanplus -U root -H
192.168.0.80 -P pass sdr type Current
```

4 Spécifications & Utilisation

4.1 Version « Démon de collecte par VM »

Nous fournissons un exécutable qui permet d'estimer la consommation d'une machine virtuelle déployée dans le *cloud*. L'exécutable doit être copié dans la VM lors de son déploiement. Quand on lance le programme, il écrit simplement sur la sortie standard la consommation instantanée estimée de la VM. L'invocation se fait automatiquement avec le mécanisme de SLA. La valeur est ainsi consommée par le module COMONS (module de monitoring).

Exemple de SLA : vérification de la consommation énergétique toutes les minutes, et envoi d'une alerte si la consommation dépasse les 80 watts.

```
<terms name="accords:guarantees" type="guarantees">
<term   name="test:guarantee1">
        <guarantee name="test:g1" obligated="provider" importance="normal"
                        scope="default"> <variable name="test:gv1" property="watt"
                                condition="less" value="80"/>
        <business name="test:bv1" nature="reward"
                expression="cordscript:packet.consume();"/>

                <business name="test:bv2"       nature="penalty"
                        expression="cordscript:alert.new();"/> </guarantee>

</term>
</terms>
```

Exemple de fichier metric.xml :

```
<metric name="watt" period="60" samples="1"   expression="coeescollector"/>
```

4.2 Collecte centralisée (PoC Mandriva)

Pour ce *Proof of Concept* (POC), nous utilisons une infrastructure matérielle de *data-center* privé (*greencloud*) déployé à l'École Normale Supérieure de Lyon. Les mesures de sécurité déployée étant très restrictive, seuls les modules fournis par l'équipe INRIA RESO permettent une extraction des données de consommation électrique. Nous présentons alors les données de consommation énergétique à travers une interface web publique en accord avec les règles de sécurité.

Une requête GET sur cette URL :

http://greencloud.ens-lyon.fr/web/compatibleone/coees/service/list/

permet de lister les services déployés ainsi que leurs états dans un format JSON.

Exemple réponse JSON :

{"ad66dbea-90cf-459b-a294-cf9120aac5a0":{"nameVM":"one-442","uptime":"1d 00:05","status":"started"}}

{"a44662da-3c13-65a3-c983-6dc1b5a5e652":{"nameVM":"one-443","uptime":"0d 00:52","status":"stopped"}}

On peut alors interroger un service sur sa consommation avec un GET sur
http://greencloud.ens-lyon.fr/web/compatibleone/coees/service/uuid

On obtiendra en format JSON les 20 dernières mesures.

Exemple avec uuid = ad66dbea-90cf-459b-a294-cf9120aac5a0 :

{ "0" : "5.36625", "10" : "5.3", "20" : "5.3", "30" : "5.3", "40" : "5.36625", "50" : "5.3", "60" : "5.36625", "70" : "11.925", "80" : "11.925", "90" : "11.925", "100" : "11.925", "110" : "11.925", "120" : "11.925", "130" : "11.925", "140" : "11.925", "150" : "11.925", "160" : "7.48625", "170" : "5.3", "180" : "5.3", "190" : "5.36625" }

Ce qui correspond à :

"0" : "5.36625" → la dernière valeur est 5.36625 watt

"10" : "5.3" → la valeur 10 secondes avant est 5.3 watt

"20" : "5.3" → la valeur 20 secondes avant est 5.3 watt

...

"80" : "11.925" → la valeur 80 secondes avant est 11.925 watt

...

"190" : "5.36625" → la valeur 190 secondes avant est 5.36625 watt

Afin de suivre l'évolution de la consommation énergétique d'une infrastructure matérielle, il est également possible de couper et de relancer les machines avec un GET sur :
http://greencloud.ens-lyon.fr/web/compatibleone/coees/service/stop.php?id=uuid
pour couper un service
et un GET sur :
http://greencloud.ens-lyon.fr/web/compatibleone/coees/service/start.php?id=uuid
pour relancer le service.

Les résultats sont visibles dans l'interface web suivante :

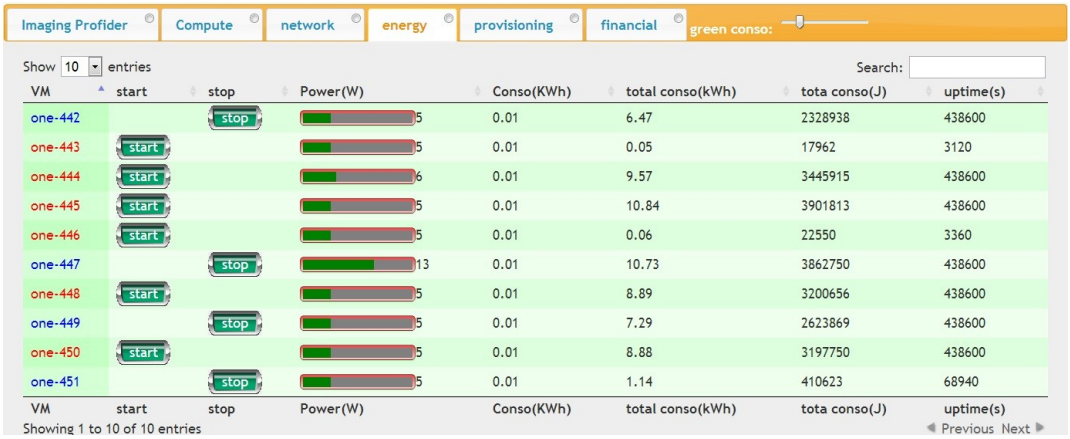

Figure 2: interface web du PoC Mandriva : visualisation des consommations énergétiques

5 Annexes

5.1 Annexe 1 – Lecture de la base BerkeleyDB

Voici un exemple de code permettant l'insertion d'une donnée dans notre base *BerkeleyDB* assurant le stockage de nos consommations. Nous insérons une consommation en watt pour un couple *timestamp* – machine donné.

Pour cela, on utilise une clé composite, qui est la concaténation du *timestamp* et *hostID*.

```
void insertDB(DBContext* ctx, int timestamp, int hostID, float value)
{
	DBT key, data;
	// Zero out the DBTs before using them.
	memset(&key, 0, sizeof(DBT));
	memset(&data, 0, sizeof(DBT));

	struct key k = {timestamp, hostID};

	key.data = &k;
	key.size = sizeof(k);

	data.data = &value;
	data.size = sizeof(value);

	int ret = ctx->dbp->put(ctx->dbp, NULL, &key, &data, 0);
	if(ret != 0)
	{
		ctx->dbp->err(ctx->dbp, ret, "Put key %f failed !
			Err mor(%d) = %s\n", timestamp,
			ret,db_strerror(ret));
	}
}
```

5.2 Annexe 2 - Association prise/machine (XML)

Fichier exemple XML de correspondance prise / machine :

```
<clusters>
<cluster name="RESO">
<probe model="eaton" ip="192.168.0.202" disable="1">
        <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 id="2" ip="10.5.5.10" outlet="18"/>
                        <host id="3" ip="10.5.5.11" outlet="19"/>
        </hosts>
</probe>
<probe model="ipmi" ip="192.168.0.70" password="copernic">
        <hosts>
                        <host id="4" ip="10.5.5.8"/>
        </hosts>
</probe>
<probe model="ipmi" ip="192.168.0.80" password="copernic">
        <hosts>
                        <host id="5" ip="10.5.5.9"/>
        </hosts>
</probe>
</cluster>
</clusters>
```

Dans cet exemple, nous avons 4 machines :
10.5.5.10, 10.5.5.11, 10.5.5.8 et 10.5.5.9

10.5.5.10 et 10.5.5.11 sont branchées sur un ePDU Eaton. Le ePDU est interrogeable via le réseau avec l'IP 192.168.0.202. 10.5.5.10 est sur la prise numéro 18 et 10.5.5.10 est sur la prise numéro 19.

Les consommation des machines 10.5.5.8 et 10.5.5.9 sont accessibles par carte IPMI aux adresses respectives : 192.168.0.70 et 192.168.0.80

Interrogation IPMI :

```
ipmitool -S ipmi192.168.0.80.cache -I lanplus -U root
-H 192.168.0.80 -P pass sdr type Current
```

Interrogation SNMP (ePDU eaton) :

```
snmpget -v 1 -c public 192.168.0.203 actualVoltageO.6.1
SPGW-MIB::actualVoltageO.6.1 = INTEGER: 232.8
snmpget -v 1 -c public 192.168.0.203 actualCurrentO.6.1
SPGW-MIB::actualCurrentO.6.1 = INTEGER: 0.4

snmpget -v 1 -c public 192.168.0.203 powerFactorO.6.1
SPGW-MIB::powerFactorO.6.1 = INTEGER: 86.9 %
```

5.3 Annexe 3 – Spoofing avec Gmetric

Le client métrique *Ganglia* (Gmetric) annonce une nouvelle métrique à tout les démon de collecte qui écoute sur le canal *multicast*. Gmetric permet d'annoncer des valeurs en se faisant passer pour l'hôte cible qui donnera l'illusion qu'il a lui même envoyée sa valeur. Ce procédé a pour but de centraliser les données via le réseau et de faire du stockage ou encore des traitements supplémentaire sur ces valeurs (consommation totale, moyenne, comparaison, etc.)

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

Le frontal web de *Ganglia*, dont certaines pages ont été adaptées pour ajouter les graphiques de consommation d'énergie, est situé sur la machine de service COEES qui a pour but d'héberger la base de données ainsi qu'un serveur web pour présenter les données de consommation avec une interface web.

Figure 3: Consommation globale du cluster INRIA RESO avec Ganglia