UNIVERSITY OF LEEDS

This is a repository copy of *PaaS-IaaS Inter-Layer Adaptation in an Energy-Aware Cloud Environment*.

White Rose Research Online URL for this paper:
http://eprints.whiterose.ac.uk/119030/

Version: Accepted Version

Article:
Djemame, K, Bosch, R, Kavanagh, R et al. (4 more authors) (2017) PaaS-IaaS Inter-Layer Adaptation in an Energy-Aware Cloud Environment. IEEE Transactions on Sustainable Computing, 2 (2). pp. 127-139. ISSN 2377-3782

https://doi.org/10.1109/TSUSC.2017.2719159

(c) 2017, IEEE. Personal use is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works. This is an author produced version of a paper published in IEEE Transactions on Sustainable Computing. Uploaded in accordance with the publisher's self-archiving policy.

Reuse
Items deposited in White Rose Research Online are protected by copyright, with all rights reserved unless indicated otherwise. They may be downloaded and/or printed for private study, or other acts as permitted by national copyright laws. The publisher or other rights holders may allow further reproduction and re-use of the full text version. This is indicated by the licence information on the White Rose Research Online record for the item.

Takedown
If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

White Rose university consortium
Universities of Leeds, Sheffield & York

eprints@whiterose.ac.uk
https://eprints.whiterose.ac.uk/

PaaS-IaaS Inter-Layer Adaptation in an Energy-Aware Cloud Environment

Karim Djemame, *Member, IEEE,* Raimon Bosch, Richard Kavanagh, Pol Alvarez, Jorge Ejarque, Jordi Guitart and Lorenzo Blasi

Abstract—Cloud computing providers resort to a variety of techniques to improve energy consumption at each level of the cloud computing stack. Most of these techniques consider resource-level energy optimisation at IaaS layer. This paper argues energy gains can be obtained by creating a cooperation between the PaaS layer (in charge of hosting the application/service) and the IaaS layer (in charge of handling the computing resources). It presents a novel method based on steering information and decision taking to trigger the PaaS and IaaS layers to adapt their energy mode in service operation, therefore enabling the Cloud stack to actively adapt to changing situations. Experimental results demonstrate such adaptation achieves dynamic energy management in each of the PaaS and IaaS cloud layers.

Index Terms—Cloud computing, Virtualization, self-adaptation, Service-level agreements, energy modelling, OVF

1 Introduction

Advances in cloud computing research have in recent years resulted in considerable commercial interest in utilising cloud infrastructures to support commercial applications and services. However, cloud computing as a leading ICT approach provides elastic and on-demand ICT infrastructures makes up a large proportion of the total ICT energy consumption. Predictions have been made on an unsustainable quadrupling in the energy consumption and carbon emissions of data centres used to operate cloud services by 2020 [1] with comparable emissions to the aeronautical industry.

Cloud computing providers resort to a variety of techniques to improve energy consumption at each level of the cloud computing stack. Therefore, research on energy efficiency in clouds has attracted considerable attention and has focused on many aspects including ICT equipment (servers, networks) as well as software solutions running on top of ICT equipment (e.g. cloud management system domain for managing the cloud infrastructure), see [2] for a survey. These solutions aim at optimising the consumed energy for running cloud services, and different techniques are used to achieve energy efficiency, e.g adjust CPU voltage and frequency according to the load for saving energy, dynamically resize active servers according to varying workload conditions, workload consolidation on a number of servers etc.

Previous work argued that research is needed to propose novel methods and develop tools to support software developers in monitoring, minimising the carbon footprint and optimising energy efficiency resulting from running services in cloud environments [3]. Therefore, the primary goal was to characterise the factors which affect energy efficiency in software development, deployment and operations. The approach firstly focuses on the identification of the missing functionalities to support energy efficiency across all cloud layers (SaaS, PaaS, IaaS), and secondly on the definition and integration of explicit measures of energy requirements into the design and development process for software to be executed on a cloud platform.

Self Adaptive Systems have seen a significant level of interest in different research areas like autonomic computing and pervasive computing [4]. They provide self-management properties and exhibit system properties such as self-awareness to achieve adaptation. They are capable of monitoring their resources, state and behaviour.

This paper focuses on the operation of cloud services through the implementation of novel methods within a reference energy-aware and self-adaptive architecture. It specifically argues energy gains could be obtained by creating a cooperation between the PaaS layer (in charge of hosting the application/service) and the IaaS layer (in charge of handling the computing resources). Adaptation requires the integration of the capabilities required to achieve dynamic energy management in each of the PaaS and IaaS cloud layers. This includes the provision of the means to assess the services' compliance during their operation to the terms of a negotiated Service Level Agreement (SLA) as well as the overall energy efficiency from the Cloud provider perspective. Steering information and decision taking to trigger the PaaS and

- *K. Djemame and R. Kavanagh are with the School of Computing, University of Leeds, UK, LS2 9JT.*
  E-mail: {K.Djemame,R.E.Kavanagh}@leeds.ac.uk
- *R. Bosch, P. Alvarez, J. Ejarque and J. Guitart are with Barcelona Supercomputing Center and Universitat Politecnica de Catalunya - Barcelona Tech., Spain.*
  Email: {raimon.bosch,pol.alvarez,jorge.ejarque,jordi.guitart}@bsc.es
- *L.Blasi is with HPE, Italy.*
  Email: {L.Blasi}@hp.com

IaaS layers to adapt their energy mode is key, therefore enabling the Cloud stack to actively adapt to changing situations. Cloud service and infrastructure providers benefit from such adaptation as it allows the deployment and operation of services in an energy-efficient aware architecture which is able to automatically adjust itself in response to changes in its operating environment.

The main contributions of this paper are:

- A detailed self-adaptive architecture that facilitates an energy aware and efficient cloud operation methodology on PaaS/IaaS;
- a PaaS-IaaS inter-layer adaptation methodology supported in such architecture;
- The results of a performance evaluation and feasibility study of the methodology implementation for the deployment and operation of cloud applications.

The remainder of the paper is organised as follows. Section 2 describes a proposed architecture to support energy-awareness. Section 3 describes self-adaptation in from the perspective of PaaS and IaaS layers, respectively. Section 4 explains PaaS-IaaS inter-layer coordination within the architecture. Section 5 presents the experimental design, and Section 6 discusses the evaluation results of inter-layer self-adaptation within the layers. Section 7 reviews the related work. In conclusion, Section 8 provides a summary of the research and plans for future work.

2 Energy-Aware Cloud Architecture

As argued in Section 1 methods and tools that consider energy efficiency are needed to manage the life cycle of cloud services from requirements to run-time through construction, deployment, operation, and their adaptive evolution over time. Their availability will result in an implementation of a software stack for energy efficient-aware Clouds. Thus, an architecture supporting energy efficiency and capable of self-adaptation while at the same time aware of the impact on other quality characteristics of the overall cloud system such as performance is proposed. Consequently, the research questions that need to be addressed in this context are the normalisation of energy measurements, the mapping between hardware, VM and software level, the management of Key Performance Indicators (KPIs) of contributing/conflicting goals as well as the identification of variability points available for (self)-adaptation.

Figures 1-3 provide an overview of the proposed architecture. It includes the high-level interactions of all components, is separated into three distinct layers and follows the standard Cloud deployment model. Next, details on the interactions of the architectural components are discussed.

2.1 Layer 1 - SaaS

In the SaaS layer a set of components interact to facilitate the modelling, design and construction of a Cloud application. The components aid in evaluating energy consumption of a Cloud application during its construction. A number of plug-ins are provided for a frontend *Integrated Development Environment* (IDE) as a means for developers to interact with components within this layer. A number of packaging components are also made available to enable provider agnostic deployment of the constructed cloud application, while also maintaining energy awareness.

The IDE is intended to be the main entry point to the infrastructure for service designers and developers. The idea is that the IDE integrates the graphical interfaces to the different tools available in the SaaS layer, thus offering a unified and integrated view to users. The

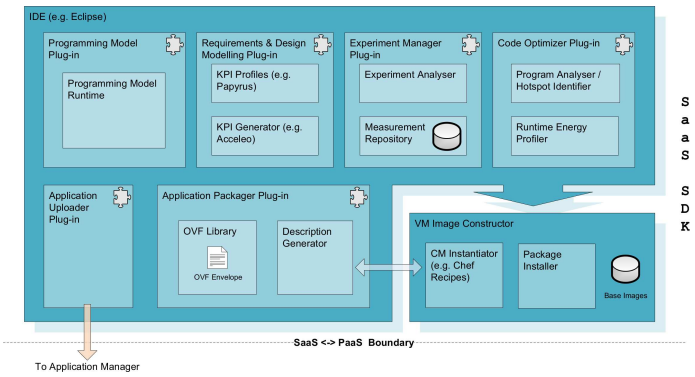

Fig. 1. Architecture - SaaS

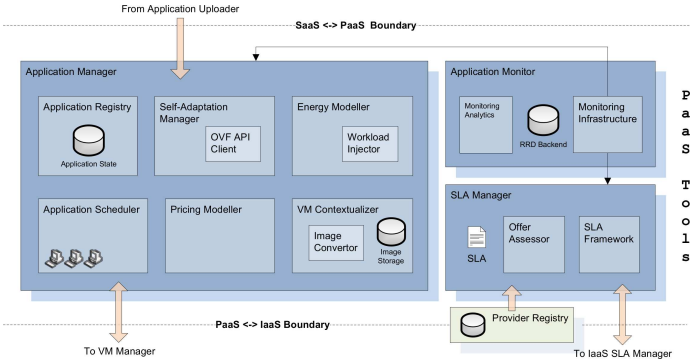

Fig. 2. Architecture - PaaS

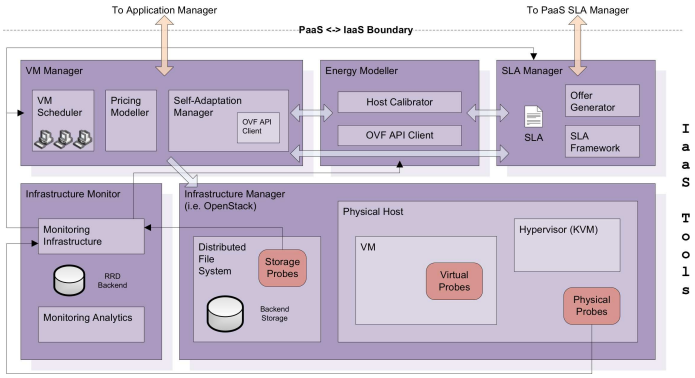

Fig. 3. Architecture - IaaS

Requirements and Design Modelling Plug-in provides developers with tools to aid in is based on the energy aware modelling of an application. The *Deployment Experiment Manager (DEM) Plug-in* is used prior to a SaaS application deployment and helps a SaaS development team to determine what deployment configuration alternatives of their SaaS application is likely to provide the most effective business operation. The *Code Optimizer* plays an essential role in the reduction of energy consumed by an application. This is achieved through the adaptation of the software development process and by providing SaaS software developers the ability to direct understand the energy foot print of the code they write. The Programming Model Plug-in is based on COMPSs [5] and provides an interface to the developer to create applications that follow the energy aware programming model [6]. Finally the Deployment Experiment Manager Plug-in, helps to associate the outputs of the SaaS Modelling tools with the workloads and the energy-aware architecture.

Other components in this layer include 1) the *Application Packager* component is in charge of packaging applications. This component takes into account input from the Requirements and Design Modelling Plug-in in OVF format to package the software with the different requirements. It also generates a Service Manifest to submit to the VM Image Constructor; 2) the *VM Image Constructor* (VMIC) uses the application packages and the service manifest or application descriptor to create VM images that can be deployed in the PaaS layer, and 3) the *Application Uploader* interacts with the PaaS Application Manager to register the final VMs ready for deployment.

2.2 Layer 2 - PaaS

The PaaS layer provides middleware functionality for a Cloud application and facilitates the deployment and operation of the application as a whole. Components within this layer are responsible for selecting the most energy appropriate provider for a given set of energy requirements and tailoring the application to the selected providers hardware environment. Application level monitoring is also accommodated for here, in addition to support for Service Level Agreement (SLA) negotiation.

The *Application Manager* (AM) component manages the user applications that are described as virtual appliances, formed by a set of VMs that are interconnected between them. The role of the *Virtual Machine Contextualizer* (VMC) is to embed software dependencies of a service into a VM image and configure these dependencies at runtime via an infrastructure agnostic contextualization mechanism. Additionally, the VMC enables the use of energy probes for the gathering of VM level energy performance metrics. The *Application Monitor* (APPM) is able to monitor the resources (CPU, memory, network ...) that are being consumed by a given application, by providing historical statistics for host and VM metrics: performance (e.g. CPU that an application is consuming during a given period of their execution) and energy (e.g. Watts that an application consumes during operation). The goal of the *Energy Modeller* is to gather and manage energy related information throughout the whole Cloud Service lifecycle and Cloud layers: from requirement level KPIs to programming model annotations down to PaaS and IaaS level measurements made through the monitoring agents present at those levels. The energy modeller provides an interface to estimate the energy cost of a PaaSs KPIs, and the provided estimations assist in the selection of the appropriate IaaS provider for running the application. The *SLA Manager* is responsible for managing SLAs at PaaS level. This requires interacting with the Application Manager, the Pricing Modeller and the IaaS SLA Manager. The Application Manager provides to the PaaS SLA Manager information to establish which terms need to be scheduled and then negotiated with the IaaS Providers. Once negotiation between PaaS SLA Manager and IaaS SLA Manager is done, the PaaS SLA Manager will request the price of the build offer to the Pricing Modeller. The goal of the *Pricing Modeller* is to provide energy-aware cost estimation related to the operation of applications on top of VMs on a specific IaaS provider. In addition, this component provides billing information. The *Self-Adaptation Manager* performs adaptation at run-time, see Section 3.

2.3 Layer 3 - IaaS

In the IaaS layer the admission, allocation and management of virtual resource are performed through the orchestration of a number of components. Energy consumption is monitored, estimated and optimized using translated PaaS level metrics. These metrics are gathered via a monitoring infrastructure and a number of software probes.

The *Virtual Machine Manager* (VMM) component is responsible for managing the complete life cycle of the virtual machines that are deployed in a specific infrastructure provider as well as managing self-adaptation thanks to the *Self-Adaptation Manager*, see Section 3. The goal of the *Energy Modeller* is to gather and manage energy related information throughout the whole Cloud Service lifecycle and Cloud layers. This components core responsibility is to provide energy usage estimates by presenting the relevant KPIs for a virtual machine deployment on the infrastructure provided. This will include cost trade off analysis based on sources such as prior experience, the application profile as defined in the SLA, which is subsequently translated into infrastructure level KPIs, and finally from current up to date monitoring information from the deployment environment. The *SLA Manager* is responsible for managing SLA negotiation requests at IaaS level. It interacts with the VM Manager to get the status of the available resources

in order to determine the SLA offer and the Pricing Modeller to assign a price to the offered terms. The goal of the *Pricing Modeller* is to provide energy-aware cost estimation related to the operation of the physical resources managed by the IaaS provider and used by specific VMs. In addition, it provides billing information. The *Infrastructure Manager* (IM) manages the physical infrastructure and redirects requests to hardware components. It maintains lists of hardware energy-meters, physical cluster nodes, network components and storage devices. External components can obtain and manipulate the state of the infrastructure through a common API that is independent of the actual hardware. The IM in the duty to provide power consumption information for each cluster node. Furthermore, the IM requires an authentication for all operations which ensures protection against attacks as well as a sufficient separation of different parties.

3 Self-Adaptation: Layer's Perspective

The paper addresses energy-efficient management of cloud resources across the entire cloud software stack. Therefore, the proposed cloud architecture needs to support self-adaptation regarding energy efficiency while at the same time being aware of the impact on other quality characteristics of the overall cloud system such as cost and performance.

Adaptation with regard to energy efficiency focuses on the addition of capabilities required to achieve dynamic energy management in each of the Cloud layers [7]. This includes the provision of the means to assess the services' compliance during their operation to the terms of a negotiated SLA (and thus their QoS and their energy efficiency) as well as the overall energy efficiency from the Cloud provider perspective. Achieving steering information and decisions among Cloud layers for triggering other layers to adapt their energy mode is key, therefore enabling the entire Cloud stack to actively adapt to changing situations. At the PaaS and IaaS layers, the main focus will be on information sharing and decision making. Fo eliciting adaptation requirements, the 5W + 1H questions are introduced as formulated in [8] and [4]:

- When to adapt?
- Why do we have to adapt?
- Where do we have to implement change?
- What kind of change is needed?
- Who has to perform adaptation?
- How is the adaptation performed?

3.1 PaaS Perspective

After construction at the SaaS layer, the application is deployed to the PaaS layer via the Application Manager where it is deployed/monitored.

The application manager queries the provider registry for a list of IaaS providers, after which a phase of negotiation occurs. The negotiation takes advantage of the energy and pricing modeller's in order to find the negotiating position of the PaaS layer. After this phase deployment occurs.

At the start of the operation phase the application manager registers its interest in VMs to be monitored. The SLA manager monitors the application for breaches in the terms of the agreement.

The principle triggers for adaptation at this layer are:

1) Application level SLA breaches: on aspects such current application power consumption and total energy consumption, current price and total spent so far.
2) IaaS Layer Adaptation Events: the PaaS layer is able to listen to the IaaS layer for adaptation events. This may have the principle action of mitigating changes in cases where adaptation has just occurred to VMs associated with the monitored application.
3) Renegotiation Events: calls by the IaaS layer to the PaaS layer to renegotiate i.e. in cases where it can provide a better service or is currently under performing.

In the event an SLA breach occurs the PaaS-SAM is notified by the SLA manager of the breach. A renegotiation with the IaaS lsayer or a redeployment of the application on a different infrastructure takes place. The PaaS-SAM takes part in the following self-adaptive steps:

1) Identify from the OVF which adaptation types are possible. An example of this would be ensuring horizontal scaling is possible given the permitted amount of VMs.
2) Get information about the run-time environment in order to further assess the plausibility of actuators e.g. if there is a limit on power consumption then it will not take action if it is told that in acting it will cause another SLA breach.
3) Choose an actuator to invoke. The potential actuators at this level are request rescheduling in the IaaS layer or scale horizontally (adding/removing VMs). The request to reschedule a VM is the means by which the PaaS layer prompts the IaaS layer to reschedule indicating that a particular VM is of concern.
4) Invoke the actuator on the application manager.

3.2 IaaS Perspective

There are two main stages during application deployment. The negotiation phase requires a notion of the current state of the infrastructure and the performance that will be obtained by a new application submission. This information is provided by scheduling and with the use of predictions that utilise the energy and pricing modellers in this layer. The second stage during deployment is used to iteratively deploy the VMs. During the service operation phase the architecture focuses on application monitoring and SLA conformance.

The principle triggers for adaptation at this layer are:

1) VM level SLA breaches: on aspects such VM power consumption and energy consumption and VM price (and performance in cases of overselling resources).
2) PaaS Layer Responses: the PaaS layer invokes adaptation on this layer, thus the IaaS layer will capture changes enforced on it by the PaaS. This may have the principle action of mitigating changes in cases where adaptation has just occurred.
3) Renegotiation Events: Calls by the PaaS layer to the IaaS layer to renegotiate.
4) Pre-Defined Intervals: Such as VM addition, deletion and at regular timed intervals.

The VM Manager/Self-Adaptation Manager take part in the following self-adaptive steps:

1) Identify which adaptation types are possible with guidance from the Energy Modeller, thus assisting it in how to adapt. An example of this would be ensuring horizontal scaling is possible given the permitted amount of VMs.
2) Choose an actuator to invoke. The potential actuators at this level are rescheduling with vertical scalability, rescheduling with live migration or renegotiation with PaaS. The latter takes place in the event of an SLA breach or if there is a change in the cost of running the application, i.e. changes in energy price.
3) Invoke the actuator.

A summary of the PaaS-IaaS adaptation dimensions is given in Table 1.

4 PaaS-IaaS Inter-Layer Adaptation: Proposed Solution

The initiator of PaaS-IaaS inter-layer adaptation is the PaaS SLAM which, following the detection of an SLA violation on the application metrics, notifies the PaaS-SAM. The latter is in charge of deciding and implementing corrective actions, e.g. a scale-up or a scale-down operation. This decision is based upon the information provided by the VMM (which is called via the Application Manager) regarding the VM free slots that are available: number of CPUs, RAM and disk available per host. Figure 4 shows the components interaction to support such process.

To implement inter-layer adaptation, the PaaS-SAM is allowed to create VMs of different sizes for the same application (i.e. modify the number of CPUs of an existing VM registered in the Application Manager) considering vertical or horizontal scalability (or both). The PaaS-EM assists in deciding the size of VMs required for the actual service.

4.1 OVF

OVF provides the functionality to programmatically generate descriptions of Cloud based applications. In the context of this research standard compliant extensions to the OVF standard are utilised to enable self-adaptation in the architecture layers through a standardised exchange of application requirements via a unified data model. The OVF product section is extended using a key value pair based system to include both constraints and rules for self-adaptation. This includes for the product section of a virtual system specifying boundary conditions for the number of VM instances, as well as the initial count of VM instances using the keys: asceticLowerBound, asceticUpperBound, asceticStartBound. The product section can also be used to specify adaptation rules, these are added as key value pairs with each rule been appended with a numeric id value. Such as the following example of rule zero: <ovf:Property ovf:key="AdaptationRuleSlaTerm_0" ovf:type="string" ovf:value="power_usage_per_app" >Each rule has various elements each indicated by separate key value pairs:

AdaptationRuleSlaTerm Name of the rule.

AdaptationRuleComparator Comparator to be use: LTE, LT, GTE, GT, EQ

AdaptationRuleResponseType Action to be performed in case the rule is triggered.

AdaptationRuleLowerBound An optional lower bound to trigger the rule, based upon the difference between the SLAs guaranteed value and the actual value measured at SLA violation.

AdaptationRuleLowerBound An optional upper bound to trigger the rule.

AdaptationRuleNotificationType Indicating if the notification was an SLA violation, a warning or another event.

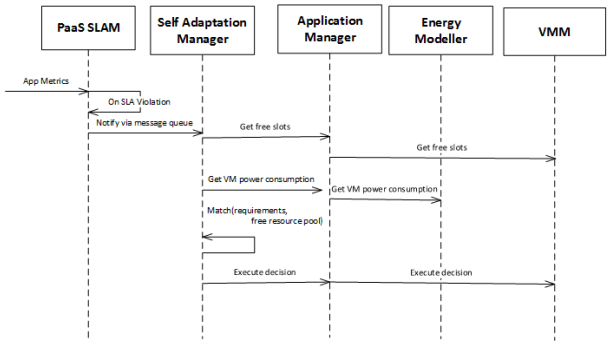

Fig. 4. Slot-Aware VM Deployment: Inter-Layer Components Interaction

4.2 SLA Management

As indicated in Figures 2 and 3, the architecture includes an SLA Manager component in each of the PaaS and IaaS layers.

Application SLA terms are created by end users in the SaaS layer, along with the OVF description of their application. SLA terms at PaaS level express constraints on application performance, its energy consumption or the price of resources. PaaS SLAM is in charge of contacting IaaS SLA Managers from different providers and

TABLE 1
PaaS-IaaS Inter-Layer Adaptation: Dimensions

Adaptation requirement	PaaS	IaaS
When to adapt?	Reactive / Proactive	Reactive / Proactive
Why to adapt?	Application SLA breach IaaS event, IaaS Renegotiation	VM SLA breach PaaS event, PaaS Renegotiation
Where to implement change?	VMs	VMs
What kind of change is needed?	Parameter adaptation: VM count, VM CPU count	Parameter adaptation: VM placement
Who performs adaptation?	PaaS SAM (leads) / IaaS SAM	IaaS SAM (leads) / PaaS SAM
How is the adaptation performed?	VMs rescheduling VM scalability Redeployment to another provider	VMs rescheduling VMs scalability
Objective	Minimise power Minimise energy	Minimise power Minimise energy

negotiate the resources required to run the application developed at the SaaS level. PaaS SLAM negotiates with multiple IaaS providers, comparing multiple offers and selecting the best one. A contract is produced in the form of a list of SLA terms, including the computational resources agreed, their maximum energy consumption and the price that the application will pay for using them. SLA thresholds associated with SLA terms are monitored and in case of a breach an SLA violation is raised to the SAM (in case of PaaS) and the VMM (in case of IaaS) to take corrective actions derived from OVF.

IaaS and PaaS SLAMs both allow renegotiation of existing SLA agreements. For this, an updated SLA template with new thresholds is provided and both parties negotiate until they reach a new agreement. The original SLA agreement is still valid during the renegotiation and, in case the renegotiation succeeds, the new agreement supersedes the original one. The renegotiation takes place while the resources (VM) remain up and running, so that the service is not interrupted.

4.3 Energy Modelling

The PaaS Energy Modeller (EM) collects measurements generated by applications in order to build their energy models. PaaS EM provides energy measurements for both an application and the events it generates. Application events, defined through the SaaS tools, are monitored through application probes running inside Virtual Machines.

PaaS EM not only calculates energy measurements, but also provides an API to estimate future consumption of an application and its events. Such API, when invoked, builds an energy model trained with historical data collected from the running application. In particular, for each VM within the same deployment, a power model is generated using historical data. When an estimation is required, the model is applied to forecast the future consumption trend. If the estimation is requested for the whole application, this calculation is applied to each VM within the same application deployment and each predicted power is summed together to provide the predicted power for the application as a whole.

Forecasting of future values is done using Neural Network models. PaaS EM uses the collected power utilization measurements of each application's VM to train a model for that VM. In order to forecast consumption, PaaS EM uses the Neural Network on a sliding window of input values. At each step a new value is produced and added as the last input value of the next step. The future value is produced using enough steps to reach the desired instant. The implementation is based on Neuroph [9], which is an object-oriented neural network framework written in Java. Neuroph provides a Java class library and can be used to create and train neural networks, both in Java programs and using a graphical user interface (NeurophStudio). Further details on the Energy Modeller are found in [10].

4.4 Self-Adaptation Management

Self-Adaptation following the layer perspective in Section 3 is co-ordinated between separate adaptation managers in the PaaS and the IaaS layers, namely the PaaS Self-Adaptation manager (PaaS SAM) and the IaaS SAM, the later being a module of the VMM. This adaptation considers: energy, performance and cost of the adaptation it performs.

The first stage of an adaptation is a notification event from the PaaS SLAM (see Figure 4), this can be either an SLA breach, a warning of an impending breach or a notification of other events. Notifications of SLA breaches principally contain the following information:

Time: the timestamp of the detected violation.

Value: a raw value representing how large the breach is, i.e. the measured value of the violation.

Type of violation message: This is either a "violation" if the violation is detected, a "warning" if the guarantee is near the violation threshold, or an informative indicator to state a new scaling time period has been reached where the application may be rescaled.

SLA UUID: the UUID of the SLA.

SLA Agreement Term: used to distinguish between different constraint terms.

SLA Guaranteed State: Provides information on the border conditions of the SLA:

Guarantee Id: it is the metric to be monitored.
Operator: such as greater than, less than, equal.
Guaranteed Value: the value of the threshold.

On notification the PaaS SAM decides upon the course of action to take, which is principally made up of adaptation rules and actuators. The adaptation rules consider the mapping between the type of notification and the adaptation to perform. The adaptation phase works in two stages. The first phase indicates the type of adaptation to make such as: add/remove VMs by assessing the causes of the SLA breach, these are covered by the adaptation rules. The second phase indicates the exact nature of this adaptation such as what type of VM to add or which VM should be deleted. This considers aspects such as pre-agreed permissible scaling, the cost, power and energy consumption of the overall application.

In the first phase that utilises adaptation rules that can be specified from OVF (see section 4.1) the rule is recorded as a tuple of:

```
<Agreement Term, Comparator, Response Type
{Event Type}, {Lower Bound}, {Upper Bound},
{Parameters}>
```

which is utilised to determine the form of adaptation to take. Two examples of this are:

```
<energy_usage_per_app, LT, REMOVE_VM>

<power_usage_per_app, LT, REMOVE_VM,
WARNING, 0, 100, VMType = "JBoss">
```

The latter optional values allow for stronger granularity and may be loaded in as generic rules or on a per application basis as loaded from application specific OVF. This provides greater flexibility to do things such as:

- Responding to warnings, in a different fashion to SLA breaches or informative notifications.
- Observing the difference between the guaranteed value and the measured value and providing a stronger response if the deviation is further away (i.e. the lower bound and upper bound values).
- Parametrising the rules, so applications can better indicate which VM types to adapt. This allows information events such as ”it is 6pm” to result in the scaling back of resources to cope with the ”out of hours” workload, thus saving energy.

A threshold value, which determines how many events are required before a rule fires is also utilised, thus ensuring that temporary reporting of SLA breaches can be ignored. An example of this would be if VM power was to become too high due to a short burst of CPU utilisation.

The second phase then decides upon the scale of adaptation. This involves the usage of a decision engine that considers various parameters, such as OVF job specifications, SLA limits and the current environment to decide upon the scale of the adaptation and upon which VMs the adaptation should occur upon.

There are various options that can be used to make the decision on what action to take, in the case of adding and removing of a single VM per event, this could be done: *randomly*, based upon the VM power consumption, or based upon the *last VM created*. Even simple strategies such as these still have to consider constraints in the OVF and SLA such as the minimum and maximum VMs of each given OVF type that can exist an any one time, as well as SLA energy constraints such as total energy consumption, current power consumption, energy consumed over the last hour, or other SLA constraints such as cost and performance. Another option is the application of the *Mixed Size VM Power Ranked Decision Engine* algorithm for scaling up as shown in Figure 5.

The scaling up feature of this algorithm first obtains the list of all possible VM types that are permissible to add, by checking the minimum and maximum bounds of instances for the VM type. A preference may be specified in the OVF so that the application selects a given type of VM to scale first. If this is not specified, the VM type with the lowest average power consumption is selected, to start the scaling process. The value for the lowest power is obtained from the PaaS energy modeller. It calculates for the application’s deployment the average power consumed for the given VM type. This value is later used again as one of the criteria that determines scaling of the VMs. To ensure too many VMs are not added the likely new power consumption is estimated by taking the total measured power for the deployment and average power for the VM type $\times$ count of VMs to add, ensuring this value is not above the maximum SLA bound for power consumption. The new VMs are created once it is determined that no SLA breaches will be caused by the algorithm.

This algorithm handles VMs which in the OVF specify a range of possible CPU configuration where the free slots in the IaaS layer are considered (line 8 of the algorithm, Figure 5). A simple consolidation method is used with the aim of creating enough VMs, while avoiding starting new physical hosts where possible. The scaling down functionality works in a similar fashion by removing VMs until the SLA criteria are met again, based upon how many VMs would need to be removed given the average power consumption of an OVF VM type.

The consolidation method (Figure 5) which calculates the free slots in the IaaS layer helps advise the PaaS SAM. It is responsible for calculating the combination of VMs that will lead to a consolidated solution. For that, it has to take information from PaaS-EM to calculate the estimated energy cost of a VM in its different sizes. It then queries the VMM to get information about the size of the available slots on each host. According to these two inputs, it calculates a deployment plan by looking for a combination of VMs that will fit in the slots provided by VMM while minimizing the number

```
1: procedure DECIDE(Response response)
2:     VMTypesPossibleToAdd[] ← calculateVMsPossibleToAdd(OVF.VmType.MinMaxBounds, VmType.count())
3:     TypeOfVMToAdd ← adaptationRules.getPreference()
4:
5:     if TypeOfVMToAdd == null then                ▷ null in case it is not specified in the adaptation rules
6:         TypeOfVMToAdd ← VmType[].getLowestAveragePowerConsumption()
7:     end if
8:     VmsToAdd = getConsolidatedSlots(Free slots on host, cpusRequired, reqs.MinCpusPerVM,
   reqs.MaxCpusPerVM, reqs.getRamMb(), reqs.getDiskGb());
9:     Ensure VmsToAdd does not exceed Max VM count from OVF.
10:    while SLA Conditions are not met do          ▷ such as SLA power consumption values are not exceeded
11:        VmsToAdd = VmsToAdd - 1;
12:    end while
13:    Launch new VMs (vmsToAdd)
14:    return result
15: end procedure
```

Fig. 5. Algorithm 1: Mixed Size VM Power Ranked Decision Engine algorithm

of running hosts. Given a number of CPUs to be added, it calculates the number of VMs to create and where to deploy them.

The main responsibility of the VMM is to manage the available pool of hosts by scheduling VMs to maximize host utilization and therefore power off unused hosts. The VMM collaboratively discloses, on request by the Application Manager, the list of available VM slots per physical host (in terms of available resources CPUs, RAM, and disk). This allows the PaaS-SAM to calculates a deployment plan comprising a combination of VMs that fit in the slots provided by VMM allowing the VMM to launch such a deployment plan. By fitting the new VMs in the available slots, the VMM can satisfy the PaaS request without starting new hosts.

5 Experimental Design

To evaluate the feasibility of the PaaS-IaaS inter-layer adaptation as outlined in section 4, the following presents the experimental design to test the performance of the slot-aware VM deployment scenario. Experiments are designed in the context of the energy efficient cloud architecture presented in section 2 as implemented by the ASCETiC project [11]. Their objective is to ascertain that the self-adaptation at PaaS and IaaS when monitoring a service in operation achieves dynamic energy management in each of the cloud layers. Subsection 5.1 discusses the cloud testbed used for the experimentation and the environment in which the architecture was deployed. Subsection 5.2 describes the cloud application used to test the inter-layer self-adaptation and the experimental set-up that includes a description of variables monitored.

5.1 Cloud Testbed

The cloud testbed used in experimentation is located at the *Technische Universität Berlin* (see Figure 6). The

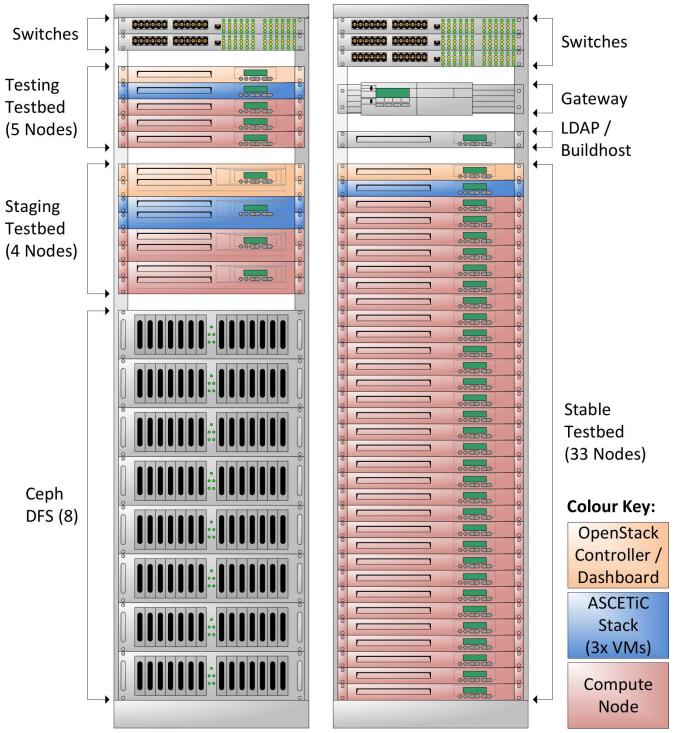

Fig. 6. Cloud Testbed

computing cluster consists of 200 commodity 1U nodes and 4 2U nodes used as a staging environment. Each of the 1U nodes is equipped with a quad-core Intel E3-1230 processor at 3.3 GHz, 16 GB of RAM, 1 TB of local hard disk capacity. The 2U nodes are equipped with 2 quad-core Intel E5620 processors at 2.66 GHz, 32 GB of RAM, 750 GB of local hard disk capacity. An 8 node Ceph cluster with a replica pool size of 2 and 16 TB of usable storage provided a Distributed File System (DFS). Additionally, the cloud testbed deploys OpenStack[12] to manage virtual infrastructure and Zabbix[13] to store monitored data. All experiments presented in this paper

were performed on the 1U nodes.

Each node is connected to two different networks and is able to transfer in duplex at full speed 1 Gbit/s. The first network is dedicated for infrastructure management via OpenStack, as well as regular data exchange between the nodes and VMs (both private and public subnets). The second network is available for storage area network usage only, with storage nodes accessible via the Ceph DFS. The ASCETiC components/tools were deployed by layer into three VMs: SaaS, PaaS and IaaS.

Power consumption on each node is measured thanks to identical energy-meters to guarantee comparative measurements. The actual devices are *Gembird EnerGenie Energy Meters* [14] that share their measurements in the local network. These devices can measure power up to 2500 watts with an accuracy of $\pm 2\%$ and are able to deliver two measurements per second.

5.2 Cloud Application

The chosen application to fulfil the objectives of the experiments is the SocialSensor [15] application that facilitates digital journalism. The application is used to identify and visualise events and trends across social media sources in real time, identify key sources and opinion formers around any event, and support journalists in verifying user generated content (text, images, video and audio) from social media sources. This application can not only demonstrate PaaS-IaaS inter-layer adaptation but is an example of how a streaming application can be implemented using the Programming model as well.

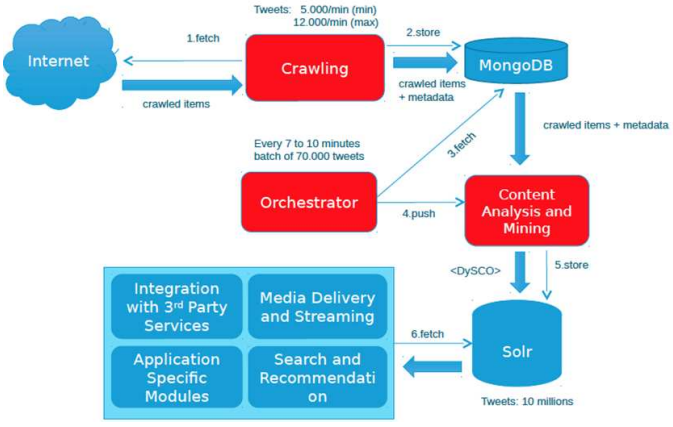

Fig. 7. Architecture of SocialSensor Application

Figure 7 describes SocialSensor application architecture. The Crawling component is a stream manager which is in charge of gathering information from social media sources and its storage in a noSQL database. The orchestrator component is in charge of periodically processing the crawled items, performing a content analysis and storing the results in a Solr database which makes results available to third party services and users. The Orchestrator component processes bags of around 70,000 items (depending on time and social media publication rate). From these items, the Orchestrator creates a list of DySCOs (Dynamic Social COntainer), an abstraction element to organise multimedia content, and divides it in three sub-lists to be processed by enriching and updating DySCOs, extracting keywords and combining trends. Finally the processed DySCOs are inserted into the Solr database.

The application was cloudified by deploying one VM per component: Crawling, Orchestrator (including Content Analysis), MongoDB and Solr databases. In the original implementation, the Orchestration element was implemented with a loop which periodically got the crawled items for processing. This implementation performed fine with a constant small set of crawled items. However, when the items rate increased, the orchestrator was not able to process all the items and therefore this led to loosing the real-time behaviour of the content analysis. For this reason, the Orchestrator component implementation was ported with the energy-aware Programming Model [6] which is based on COMP Superscalar[16]. It provides a task-based programming model to parallelise and distribute the computation in different cloud computing resources. In this case, various parts of the processing performed by the Orchestrator were defined as tasks. During the Orchestrator component execution, the programming model runtime detects the task invocation and creates a Task Dependency Graph. The tasks which are free of dependencies are executed in the available computing nodes. Moreover the runtime continuously monitors the application load (tasks pending to execute) and automatically adapts the number of the available VMs to this load by contacting the PaaS Application Manager to deploy and destroy VMs the assigned to the application.

6 Evaluation

The following section discusses the performance of the PaaS-IaaS inter-layer adaptation presenting an analysis of the experimental results. It demonstrates the PaaS-SAM ability to cater the scaling of VMs according to their power consumption under various scenarios. The experimental results are discussed in the context of the SocialSensor application.

6.1 PaaS Self-Adaptation Trigger

Figure 8 demonstrates using the Social Sensor application the PaaS SAM's ability to cater for scaling VMs based upon the expected workload. This is shown by submitting 2 separate events indicating changes in the expected workload. This gives rise to three clear stages in the graph. The 1st stage in the graph considers a normal workload in which two orchestrator and a crawling VMs are running. The 2nd stage results in orchestrator 1 terminating, and is induced by the arrival to the PaaS SAM of a low_workload_period event. The final stage is where a high_workload_period event arrives causing

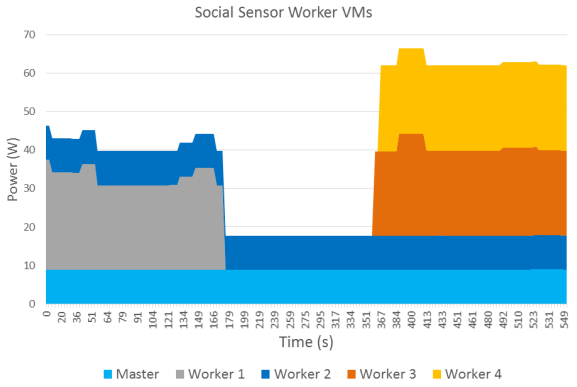

Fig. 8. PaaS Adaptation Events - Application Scaling

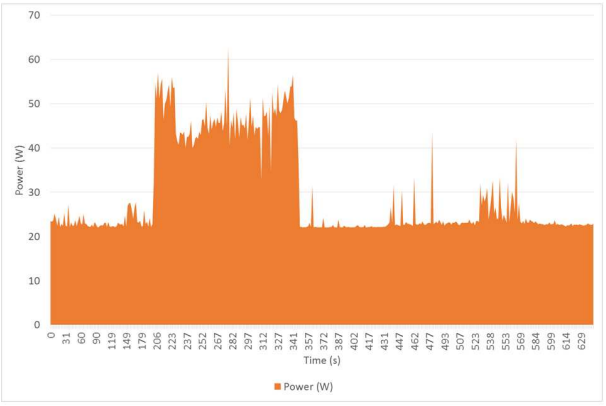

Fig. 9. VM power trace from the PaaS EM

two new orchestrator VMs to be instantiated, leaving 4 VMs in total.

The PaaS SAM queries the PaaS EM via the application manager in order to get details of a VMs power consumption. This query is conducted by passing the application and deployment ids along with the VM id. The PaaS SAM does this to obtain both the average power consumption of a VM type from the specified application deployment along with the total power usage of a deployment. This information coupled with the SLA limit on power consumption gives a strong indication of how many VMs may be added before an SLA is breached by any proposed adaptation i.e. the following must hold: sla_power_limit >total_current_power + (average_power_for_VM_type × count_of_VMs_to_add). Figure 9 shows trace of power consumption from a single ascetic-pm-socialSensorWorker VM as load is induced. The trace was generated by polling the PaaS EM using the same method utilised by the PaaS SAM. The load for the purpose of generating this trace was generated using dd on the VM that was undergoing monitoring. The information provided by invoking the PaaS EM against all VMs in the deployment allows for its total power to be calculated as well as the average power for a given VM type within the deployment. This information therefore can be used to determine how many VMs may be added before an SLA breach is likely to occur.

6.2 VMs Vertical Scaling

Since the SocialSensor application is not CPU-intensive enough stress tests are used in order to induce clear differences in power consumption. The aim is to show all VMs on a specific deployment plan set to 100% usage of CPU and consequently a high power consumption. This method will help assess the potential resources' saving in terms of power. The metrics considered across experiments are the aggregation of power consumption of the different hosts involved.

In the mixed size VM power ranked decision engine, new VMs of various size can be added to an existing deployment plan. In some situations this could lead to the usage of larger VMs than existing ones, but if it is more energy efficient to have smaller VMs then this option may be chosen. For the scale-up experiment three hosts *wally173, wally174* and *wally181* on the cloud testbed are considered, all are Intel-based with 8 CPUs each.

The experiment starts with an initial workflow of 8 CPUs: 3 CPUs are assigned to wally173 and 5 to wally174. The workload is then scaled-up to 16 CPUs. The PaaS-SAM in combination with the IaaS-SAM chooses a combination of VMs with flexible size of CPUs that avoids using the third host (wally181), making wally173 and wally174 fully utilised. Once the slot-aware deployment has taken place, the workload is scaled-down to the original situation of 8 CPUs. Again 3 CPUs are assigned to wally173, 5 CPUs are assigned to wally174, the workload is scaled-up to 16 CPUs but this time without using the option that does not avoid using a third host.

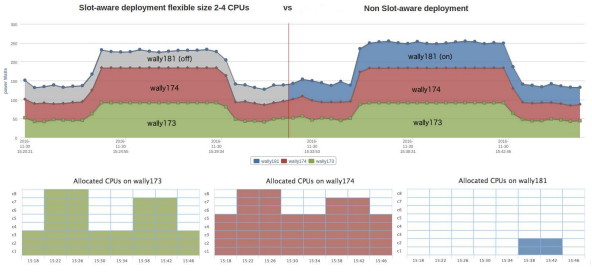

Fig. 10. VM Scale-up: Slot-Aware Deployment vs. Non Slot-Aware Deployment

The slot-aware deployment consumes less power even without considering that non-used hosts will be off as shown in Figure 10 (the slot-aware method allows for one host to be switched off). The results show how the first part of the experiment fully uses 16 CPUs on wally173 and wally174. The total power consumption when the workload is at 100% is approximately 175W.

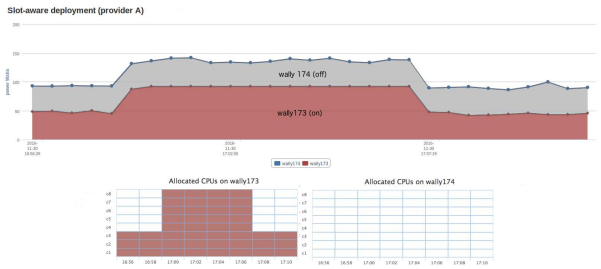

Fig. 11. Slot-Aware Deployment on Provider A Avoiding Use of wally181

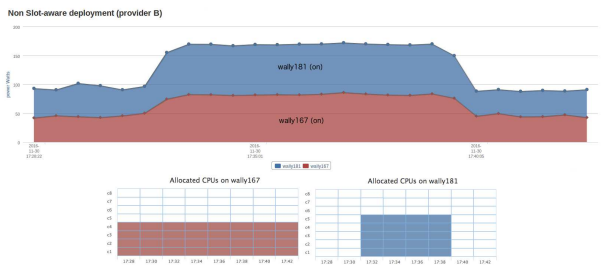

Fig. 12. Non Slot-Aware on Provider B Forcing Use of wally181

Details about the deployment plan are provided in the lower part of the Figure. In the non-slot aware deployment the increase from 8 CPUs to 16 CPUs leads to the use of 3 hosts, and therefore when the workload is at 100% power consumption is approximately 250W. The benefit ofusing the slot-aware deployment will be reduction in power consumption of 30%.

6.3 Multi-Provider Deployment

In order to test a multi-provider deployment two IaaS providers were created on the testbed, each one holding two hosts: ProviderA (wally173 and wally174), and ProviderB (wally167 and wally181). There is a slight difference in resources consumed by both providers: ProviderA has deployed one VM with 3 CPUs while ProviderB has a VM with 4 CPUs running. This means that ProviderA has one slot more than ProviderB.

When a new workload of 5 CPUs is generated, the mixed size VM decision engine chooses ProviderA as the best provider as it can fit on host wally173 without turning on wally174. After this, the workload is scaled-down to the original situation with 3 CPUs used in ProviderA and 4 CPUs in ProviderB. A new workload of 5 CPUs is generated again but this time the slot-aware deployment is not used. The workload is assigned to ProviderB where it needs to use wally167 and wally181 to complete the deployment. Therefore, wally181 must be powered on. The results of the experiment are shown in Figure 11. A new deployment of 5 CPUs will perfectly fit on wally173 at providerA (because this host is using only 3 CPUs and has 5 CPUs available). The slot-aware deployment will create a combination of VMs of 2, 3 and 4 CPUs that will fit in this slot. The total consumption when the workload is at 100% will be approximately 100W. However, should a combination of VMs be created that would choose ProviderB (see Figure 12), hosts wally167 and wally181 will inevitably be used (5 slots are needed and wally167 only has 4 available). As both hosts will need to be on the total power consumption when the workload is at 100% will be approximately 160W. In this case, the slot-aware deployment scenario represents a 37.5% of power reduction against this scenario. The experiment shows that the slot-aware deployment scenario makes an optimal use of resources in the context of multi-providers.

TABLE 2
Comparison Between the Slot-Aware vs. Non Slot-Aware Deployment with Fixed Size of 2, 3 and 4 CPUs

Slot-aware deployment	252.62 kJ
4-CPU fixed-size deployment	594.11 kJ
3-CPU fixed-size deployment	882.56 kJ
2-CPU fixed-size deployment	1,189.33 kJ

6.4 Slot-Aware VM Deployment in the Context of Full Cluster Utilisation

The final experiment aims to host a high number of CPUs and therefore considers 4 deployments with 12 CPUs each at the same time so each deployment highest peak of CPU utilisation will be 1,200% (combining 48 CPUs in total).

As shown in Table 2 the total energy consumption for the slot-aware deployment is 252.62 kJ because it only made use of 2 hosts. Deploying with a fixed-size of 4 CPUs on 3 hosts generates a consumption of 594.11 kJ, in the case of fixed-size 3 CPUs deployed in 4 hosts 882.56 kJ are consumed. Finally when deploying in groups of 2 CPUs on 6 hosts a total of 1,189.33 kJ are consumed. The results of the experiment clearly show the slot-aware deployment ability to create a combination of VMs that fits in only 2 VMs while the non slot-aware deployment makes use of many more machines. The slot-aware deployment therefore provides an optimization of the usage of the available resources.

7 Related Work

Research effort has targeted energy efficiency support at various stages of the cloud service lifecycle (construction, deployment, operation). This section reviews existing work on self-adaptation in cloud computing and categorises it into two lines of research.

The first line of work concerns self-adaptive systems in general. R. deLemos et al [17] summarized the state of-the-art and identified research challenges when developing, deploying and managing self-adaptive software systems. These challenges result from the dynamic nature of self-adaptation, which brings uncertainty. An extended architecture of the MAPE-K loop as a reference model for the design of self-adaptive systems is found in [18], assuming that the system has a central controller with a

central MAPE-K loop. The proposal consists in continuously evaluating adaptation steps concerning their actual effect and adaptation mechanisms concerning their applicability and efficiency in the case of topology changes. An agent-based modelling approach for adaptation is presented in [19]. An Agent Verification Engine (AVE) which constructs agents to perceive, react, and adapt to runtime changes of a component-based system is proposed. These agents are based on the Belief-Desire-Intention (BDI) architecture, in which agents operate in terms of motivation and beliefs. The work in [20] consolidates design knowledge of self-adaptive systems. To support software designers, the paper contributes with a set of formally specified MAPE-K templates that encode design expertise for a family of self-adaptive systems. The templates comprise: (1) behaviour specification templates for modelling the different components of a MAPE-K feedback loop (based on networks of timed automata), and (2) property specification templates that support verification of the correctness of the adaptation behaviours (based on timed computation tree logic).

The second line of work targets self-adaptation in clouds. The potential research challenges in the self-adaptation process of cloud applications in the perspective of control engineers is discussed in [21]. Considering the scenario in which the owner of multi-tenant Web application as a cloud-based application aims to use or design a controller in order to be able to satisfy the performance requirements of the users in spite of dynamic workload at runtime, the paper highlights a number of research challenges which include uncertainty, heterogeneous interfaces of cloud services and unpredictable workload. In [22], Hummaida et al. present a definition of cloud systems adaptation and a classification of key features but highlight approaches and techniques used to enable adaptation of cloud resource configuration only. A methodology for designing adaptive systems in cloud environments is proposed in [23]. It consists of several phases that take high-level stakeholders' adaptation goals and transform them into lower level MAPE-K loop control points. The MAPE-K loops are then activated at runtime using search-based algorithms. Employing the traditional MAPE algorithm: monitoring, analysis, planning, and execution, Kerstesz et al. [24] present an SLA aware architecture to deploy services in heterogeneous Clouds, with the possibility of monitoring those SLA violations and take a series of adaptive actions. Maurer et al. present a rule based approach to adapt the usage of resources in a Cloud environment at different layers [25]. Hussin et al [26] propose a Reinforcement Learning (RL) based methodology in conjunction with neural networks to design a resource scheduler that effectively observes and adapts to dynamic changes in distributed environments. A close work is the position paper by Carpen-Amarie et al [27] which argues there is a need for coordinated actions between the PaaS and IaaS layers in designing virtualised environments for energy efficiency. However, neither an implementation nor an evaluation in the context of the proposed API suite by extending LibCloud has been performed.

8 Conclusion

This paper has highlighted the importance of providing a novel methodology and tools to optimise energy efficiency in an energy-aware cloud architecture.

The approach is the extension of the capabilities of dynamic energy management by ensuring that the PaaS and IaaS cloud layers interact and cooperate to mitigate energy consumption. Such inter-layer self-adaptation ensures that the layers cooperate in order to achieve greater energy reductions than a per-layer approach can achieve on its own. Key aspects include communication and (re)negotiation between the layers so that optimisations can meet the application's requirements.

The PaaS Self-Adaptation Manager takes decisions on the type of adaptation to make as well as performs application tailored adaptation based on rule specification. The Energy Modeller forecasts power using neural networks, and estimates VM consumption based only on VM-observed data. The VM Manager manages the available pool of hosts by scheduling VMs to maximize host utilization and therefore power off unused hosts. The paper has considered a scenario where the IaaS collaboratively discloses, on request by the PaaS, the list of available VM slots per physical host (in terms of available resources vCPUs, RAM, and disk). The PaaS-SAM calculates a deployment plan comprising a combination of VMs that fit in the provided slots and requests the launch of such deployment plan. By fitting the new VMs in the available slots, the IaaS satisfies the PaaS request without starting new hosts.

The PaaS-IaaS inter-layer adaptation was showcased in the SocialSensor application that facilitates digital journalism, with the consideration of the slot-aware VM deployment. The PaaS-IaaS coordinated actions are shown to be effective through the experimental evaluation of their implementation which is already integrated in a cloud computing toolkit.

In all experiments we have seen how we gain benefits thanks to the slot-aware deployment approach. In situations with providers of 2-3 hosts we have demonstrated how it is possible to keep turned off at least one of them. Each host that is off can represent an energy saving of around 50/100W, leading to a total energy reduction of approximately 30%-37.5% per provider. We also have demonstrated that in a context of full cluster utilization the slot-aware approach performs reasonably well and the gains in energy can reach 70% when compared with the worse use-case scenarios. Another important point is that even with an architecture that does not support to turn off unused hosts, we see acceptable gains in total consumption. Therefore, placing VMs in the busier hosts by adapting their size is a good solution to consume less energy.

Future work will include exploring new approaches for PaaS-IaaS adaptation for the purpose of coordinating

the layers of the architecture presented in this paper to further increase cloud application energy efficiency. These include model-based, control theory and learning approaches. Moreover, the deployment of the architecture in a heterogeneous hardware environment will be investigated [7].

ACKNOWLEDGMENTS

The authors would like to thank the European Commission for supporting this work under FP7-ICT-2013.1.2 contract 610874 (ASCETiC project), the Ministry of Science and Technology of Spain under contract TIN2015-65316-P and the Generalitat de Catalunya under contract 2014-SGR-1051. Thanks to ATC Greece for providing the SocialSensor application and TU Berlin for their technical support.

REFERENCES

[1] M. Pawlish, A. S. Varde, and S. A. Robila, "Cloud Computing for Environment-friendly Data Centers," in *Proceedings of the Fourth International Workshop on Cloud Data Management*, ser. CloudDB '12. New York, NY, USA: ACM, 2012, pp. 43–48. [Online]. Available: http://doi.acm.org/10.1145/2390021.2390030

[2] T. Mastelic, A. Oleksiak, H. Claussen, I. Brandic, J.-M. Pierson, and A. V. Vasilakos, "Cloud computing: Survey on energy efficiency," *ACM Comput. Surv.*, vol. 47, no. 2, pp. 33:1–33:36, Dec. 2014. [Online]. Available: http://doi.acm.org/10.1145/2656204

[3] K. Djemame, D. Armstrong, R. E. Kavanagh, and et al, "Energy efficiency embedded service lifecycle: Towards an energy efficient cloud computing architecture," in *Proceedings of the Energy Efficient Systems (EES'2014) Workshop, 2nd International Conference on ICT for Sustainability 2014*, vol. 1203. Stockholm, Sweden: CEUR Workshop Proceedings, Aug 2014, p. 1–6, http://ceur-ws.org/Vol-1203/EES-paper1.pdf.

[4] C. Krupitzer, F. M. Roth, S. VanSyckel, G. Schiele, and C. Becker, "A survey on engineering approaches for self-adaptive systems," *Pervasive and Mobile Computing*, vol. 17, pp. 184 – 206, 2015.

[5] F. Lordan, E. Tejedor, J. Ejarque, R. Rafanell, J. Alvarez, F. Marozzo, D. Lezzi, R. Sirvent, D. Talia, and R. Badia, "Servicess: An interoperable programming framework for the cloud," *Journal of Grid Computing*, pp. 1–25, Sep. 2013.

[6] F. Lordan, J. Ejarque, R. Sirvent, and R. M. Badia, "Energy-aware programming model for distributed infrastructures," in *Proceedings of the 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP 2016)*, Heraklion, Greece, Feb 2016, pp. 413–417.

[7] K. Djemame, D. Armstrong, R. Kavanagh, J. Deprez, A. Ferrer, D. Perez, R. Badia, R. Sirvent, J. Ejarque, and Y. Georgiou, "Tango: Transparent heterogeneous hardware architecture deployment for energy gain in operation," in *Proceedings of the First Workshop on Program Transformation for Programmability in Heterogeneous Architectures*, S. Tamarit, G. Vigueras, M. Carro, and J. Marino, Eds., Barcelona, Spain, March 2016, http://arxiv.org/pdf/1603.01407.

[8] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 4, no. 2, p. 14, 2009.

[9] Neuroph, "Java neural network framework," 2016, http://neuroph.sourceforge.net/.

[10] A. Consortium, "Deliverable d5.1 inter-layer cloud stack adaptation," July 2016), note="http://ascetic-project.eu/content/inter-layer-cloud-stack-adaptation",.

[11] ASCETiC, "Adapting Service lifeCycle towards EfficienT Clouds," 2016, http://www.ascetic.eu/.

[12] "OpenStack: Open source software for building private and public clouds." http://www.openstack.org/, Jan 2015. [Online]. Available: http://www.openstack.org/

[13] "Zabbix - An Enterprise-class Monitoring Solution," http://www.zabbix.com/, 2016. [Online]. Available: http://www.zabbix.com/

[14] GEMBIRD Deutschland GmbH, "EGM-PWM-LAN data sheet," http://gmb.nl/Repository/6736/EGM-PWM-LAN_manual—7f3db9f9-65f1-4508-a986-90915709e544.pdf, 2013.

[15] SocialSensor, "SocialSensor," 2014, http://www.socialsensor.eu/.

[16] R. M. Badia, J. Conejero, C. Diaz, J. Ejarque, D. Lezzi, F. Lordan, C. Ramon-Cortes, and R. Sirvent, "Comp superscalar, an interoperable programming framework," *SoftwareX*, vol. 3, pp. 32–36, 2015.

[17] R. de Lemos, H. Giese, H. A. Müller, M. Shaw, and J. Andersson, *Software Engineering for Self-Adaptive Systems: A Second Research Roadmap*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–32.

[18] V. Klös, T. Göthel, and S. Glesner, "Adaptive knowledge bases in self-adaptive system design," in *41st Euromicro Conference on Software Engineering and Advanced Applications*, Funchal, Madeira, Portugal, Aug 2015, pp. 472–478.

[19] K. Johnson, R. Sinha, R. Calinescu, and J. Ruan, "A multi-agent framework for dependable adaptation of evolving system architectures," in *41st Euromicro Conference on Software Engineering and Advanced Applications*, Funchal, Madeira, Portugal, Aug 2015, pp. 159–166.

[20] D. G. D. L. Iglesia and D. Weyns, "Mape-k formal templates to rigorously design behaviors for self-adaptive systems," *ACM Transactions on Autonomous Adaptive Systems*, vol. 10, no. 3, pp. 15:1–15:31, Sep. 2015.

[21] S. Farokhi, P. Jamshidi, I. Brandic, and E. Elmroth, "Self-adaptation challenges for cloud-based applications : A control theoretic perspective," in *10th International Workshop on Feedback Computing (Feedback Computing 2015)*. ACM, 2015.

[22] A. R. Hummaida, N. W. Paton, and R. Sakellariou, "Adaptation in cloud resource configuration: a survey," *Journal of Cloud Computing*, vol. 5, no. 1, pp. 1–16, 2016. [Online]. Available: http://dx.doi.org/10.1186/s13677-016-0057-9

[23] P. Zoghi, M. Shtern, M. Litoiu, and H. Ghanbari, "Designing adaptive applications deployed on cloud environments," *ACM Transactions on Autonomous Adaptive Systems*, vol. 10, no. 4, pp. 25:1–25:26, Jan. 2016.

[24] A. Kertesz, G. Kecskemeti, and I. Brandic, "An interoperable and self-adaptive approach for sla-based service virtualization in heterogeneous cloud environments," *Future Generation Computer Systems*, vol. 32, pp. 54 – 68, 2014.

[25] M. Maurer, I. Brandic, and R. Sakellariou, "Adaptive resource configuration for cloud infrastructure management," *Future Generation Computer Systems*, vol. 29, no. 2, pp. 472 – 487, 2013.

[26] M. Hussin, N. Asilah Wati Abdul Hamid, and K. A. Kasmiran, "Improving reliability in resource management through adaptive reinforcement learning for distributed systems," *J. Parallel Distrib. Comput.*, vol. 75, no. C, pp. 93–100, Jan. 2015. [Online]. Available: http://dx.doi.org/10.1016/j.jpdc.2014.10.001

[27] A. Carpen-Amarie, D. Dib, A.-C. Orgerie, and G. Pierre, "Towards energy-aware iaas-paas co-design." in *SMARTGREENS*, M. Helfert, K.-H. Krempels, and B. Donnellan, Eds. SciTePress, 2014, pp. 203–208.

Karim Djemame Karim Djemame was awarded a Ph.D. at the University of Glasgow, UK, in 1999, and is currently holding a Senior Lecturer position at the School of Computing, University of Leeds. He sits on a number of international programme committees for cloud middleware, computer networks and performance evaluation. He was the investigator of various e-Science/Cloud projects including DAME, BROADEN, AssessGrid, ISQoS, STRAPP and OPTIMIS. He is currently involved in various research projects including ASCETiC and TANGO. His main research areas focus on Grid/Cloud computing, including system architectures, resource management, and energy efficiency. Dr. Djemame is a member of the IEEE.

Raimon Bosch Raimon Bosch is a software engineer at Barcelona Supercomputing Center, Barcelona, Spain, with a wide range of activities for over 10 years in research and development of IT systems. His expertise fields are green computing, search engine optimization and web development. Also, he has experience as entrepreneur working in early-stage start-ups and leading his own projects.

Richard Kavanagh Richard Kavanagh was awarded a Ph.D. in 2013 and is currently a research fellow at the School of Computing at the University of Leeds. He has experience working in a number of EC-funded projects including OPTIMIS, ASCETiC and TANGO. His research is in the field of Distributed Systems and the complementary paradigms of Grid and Cloud Computing, with a specific interest in quality of service, energy efficiency and resource management.

Pol Alvarez Pol Alvarez is a software developer at Barcelona Supercomputing Center, Barcelona, Spain, with a range of activities in research and development of IT systems, including workflows and distributed computing.

Jorge Ejarque Jorge Ejarque holds a PhD in Computer Architecture (2015), an Msc. in Computer Architecture Network and System (2009) and an engineering degree on Telecommunications (2005) at the Technical University of Catalunya (UPC). In 2005, he worked as IT consultant in Better Consulting then joined the Grid Computing group at BSC. During his career at the BSC, he has contributed in the design and development of different tools and programming models for HPC in distributed platforms and he currently is the product manager of the COMP Superacalar framework. He was involved in several National and International R&D projects (CoreGRID, BeInGrid, BREIN, NUBA, OPTIMIS and ASCETiC), and a member of the experts' board of the Spanish National Grid Initiative. His current research interests are focused on three areas: semantic interoperability of distributed computing platforms, parallel programming models for distributed platforms; and energy-efficient execution of distributed applications. Regarding this topic, he is currently working on two EU funded project TANGO and Euroserver.

Jordi Guitart Jordi Guitart received the M.S. and Ph.D. degrees in Computer Science at the Universitat Politecnica de Catalunya (UPC), in 1999 and 2005, respectively. Currently, he is an associate professor at the Computer Architecture Department of the UPC and an associate researcher at Barcelona Supercomputing Center (BSC), where he leads the Energy-Aware Computing area within the Autonomic Systems and e-Business Platforms group. His research interests are oriented towards green computing and the smart management of resources in virtualized datacenters. He is involved in a number of EU and industrial R&D projects.

Lorenzo Blasi Lorenzo Blasi received his Master in Electronics Engineering in 1990 and works as Technical Consultant for Hewlett Packard Enterprise (and before Digital, Compaq and HP) since 1990. His track record lists 7 years in Corporate R&D, 5 years as Software Architect for the financial market, 4 years as Business Analysis team leader for a major telecom operator, and 10 years as technical contributor to EC funded research projects. His research interests are on the following areas: Languages, Security, Cloud, SLA, Robotics. Lorenzo contributes to the TPC of several conferences, e.g. SecureWare, SECOTS (Security in Collaboration Technologies and Systems) and DIHC (Dependability and Interoperability in Heterogeneous Clouds). Within HPE Italy Innovation Center, Lorenzo is currently contributing to and owning technical responsibility for HPE Italy participation in several EC funded FP7 projects including Contrail, where he served as leader for the WP on SLAs, and currently ASCETIC, Coco Cloud and DECIDE. Lorenzo contributed to ASCETIC on the topics of Energy Modelling, SLA Management and functional analysis of Inter-layer adaptation.