Energy-Aware Pricing Within Cloud Environments

Alexandros Kostopoulos(✉), Eleni Agiatzidou, and Antonis Dimakis

Network Economics and Services Research Group, Department of Informatics,
Athens University of Economics and Business,
76 Patission Street, 10434 Athens, Greece
{alexkosto, agiatzidou, dimakis}@aueb.gr

Abstract. The Adapting Service lifeCycle towards EfficienT Clouds (ASCETiC) project aims to provide novel methods and tools to support software developers aiming to optimize energy efficiency resulting from designing, developing, deploying and running software at the different layers of the cloud stack architecture, while maintaining other quality aspects of software to meet the agreed levels. The *Pricing Modeler* is a component within the ASCETiC architecture, which is responsible for the price estimation and billing of cloud applications or Virtual Machines (VMs) based on their energy consumption. In this paper, we propose a set of novel energy-aware pricing schemes implemented within the Pricing Modeler component, as well as a set of envisaged service plans which aim to facilitate the gradual adoption of the ASCETiC architecture.

Keywords: Cloud economics · Pricing · Energy efficiency

1 Introduction

The ASCETiC project [1] complements cloud computing developments by addressing the energy efficiency of the software, which runs on cloud infrastructures. Although energy use is of relevance across all software development phases from design and implementation, we make specific reference to energy used during cloud-based service operations. The emergence of cloud computing with its emphasis on shared software components which are likely to be used and reused many times in many different applications makes it imperative for cloud service software to be developed in the most energy-efficient and eco-aware manner.

The ASCETiC approach focuses firstly on the identification of the missing functionality to support energy efficiency across all cloud layers and secondly on the definition and integration of explicit measures of energy and ecological requirements into the design and development process for software to be executed on a cloud platform.

Our main goal is to characterize the factors, which affect energy efficiency in software development, deployment and operations. The main novel contribution is the incorporation of an approach that combines energy-awareness related to cloud environments with the principles of requirements engineering and design modelling for self-adaptive software-intensive systems. This way, the energy efficiency of both cloud

J.Á. Bañares et al. (Eds.): GECON 2016, LNCS 10382, pp. 144–159, 2017.
DOI: 10.1007/978-3-319-61920-0_11

infrastructure and software is taken into consideration in the cloud service development and operation lifecycle.

Therefore, ASCETiC addresses the total characterization of software energy with respect to the impact of the software structure on energy use, which is not incorporated into any current models. It is this gap in the research agenda, which ASCETiC addresses. Determining the relationship between software structure and its energy use allows the definition of a set of software energy metrics similar in concept to those for hardware. By associating these metrics with software components and libraries, it is possible to populate a software development framework with information to predict the energy requirements of applications, thereby allowing alternative selections of software components to be made, using energy as a selection criterion.

The proposed architecture measures how software systems actually use cloud resources, with the goal of optimizing consumption of these resources. In this way, the awareness of the amount of energy needed by software will help in learning how to target software optimization where it provides the greatest energy returns. To do so, all three layers in cloud computing, namely Software, Platform and Infrastructure, will implement a MAPE (*Monitor, Analyse, Plan and Execute*) loop. Each layer monitors relevant energy efficiency status information locally and shares this with the other layers, assesses its current energy status and forecasts future energy consumption as needed. Actions can then be decided and executed according to this assessment. Hence, ASCETiC intends to make significant contributions to software engineering, programming models and adaptive architectures for clouds.

One solution for accomplishing energy efficiency could be the adoption of energy-aware pricing by the cloud service providers. Charging cloud services based on energy, will provide the necessary incentives to the customers for achieving a more efficient resource usage. In response to this challenge, the *Pricing Modeler* component, incorporated within the ASCETiC architecture, is responsible for providing energy-aware price estimation and billing related to the operation of applications or VMs associated with them.

In this paper, we propose novel pricing schemes and charging services based on actual consumption and energy efficiency of cloud resources. The energy models and the real-time monitoring mechanisms and measurements by ASCETiC make possible the creation of new pricing schemes that will charge users based on their actual consumption and energy efficiency of cloud resources. Our aim is to adapt existing pricing schemes, as well as develop new ones, thus creating an energy-efficient and at the same time economically sustainable ecosystem.

The paper is organized as follows. Section 2 gives a brief overview of the three-layer ASCETiC architecture. Section 3 provides a cloud market analysis with respect to the pricing schemes adopted by the cloud providers. In Sect. 4, we propose a set of novel energy-aware pricing schemes implemented within our Pricing Modeler component. Section 5 introduces a set of envisaged service plans intending to facilitate the gradual adoption of the ASCETiC architecture. We conclude our remarks and present our future work in Sect. 6.

2 ASCETiC Architecture

In this Section, we provide an overview of the ASCETIC Architecture. It complies with the standard cloud architecture [2] and considers the classical Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS) layers, supporting a wide range of components, including the Pricing Modeler component.

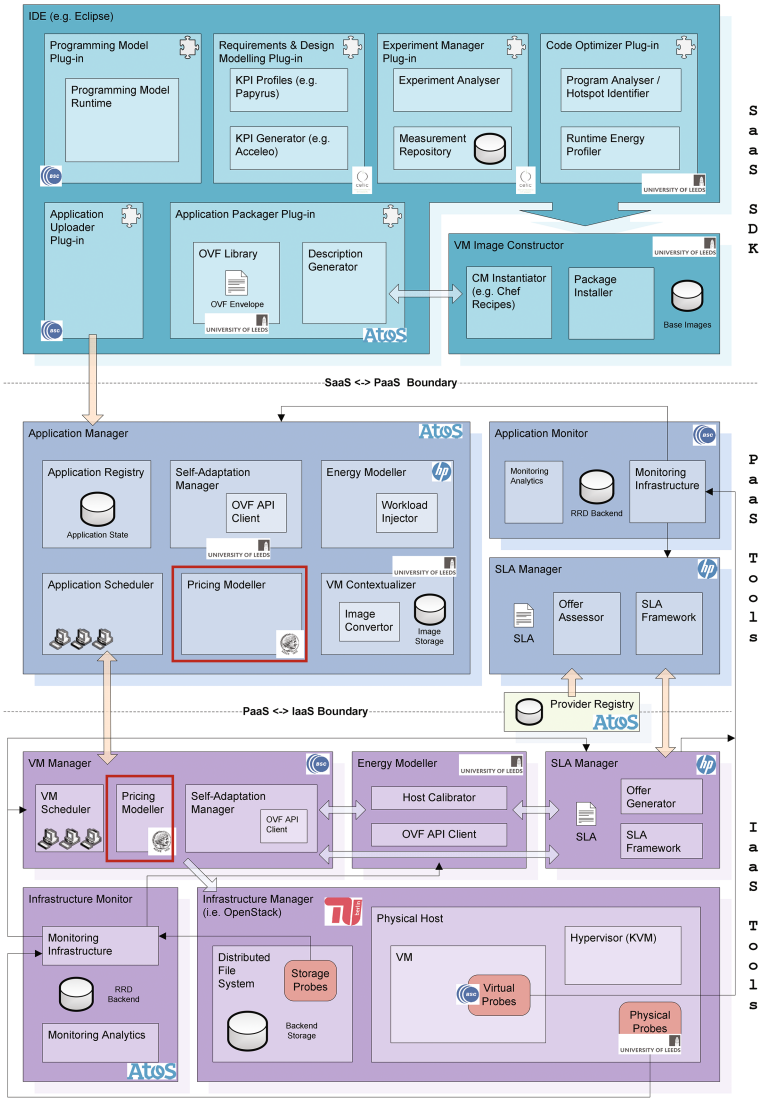

Fig. 1. Overview of the ASCETIC architecture.

In the SaaS Software Development Kit (SDK) layer, a collection of components interact to facilitate the modelling, design and construction of a cloud application. The components assist in evaluating energy consumption of a cloud application during its constructions. 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. Lastly, a number of packaging components are made available that enable provider-agnostic deployment of the constructed cloud application, while also maintaining energy awareness.

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 provider's hardware environment.

Finally, in the IaaS layer, the admission, allocation and management of virtual resources 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.

Figure 1 provides an overview of the ASCETiC architecture [3]. It includes the high-level interactions of all components, is separated into three distinct layers and follows the standard cloud deployment model.

A fully functional architecture demands the existence of a component that is dedicated to support the financial operations of the provider. Such a component focuses on the *cost function* and the *pricing schemes* of the cloud provider. Hence, as part of the presented architecture, we implement two different Pricing Modeler components.

The *PaaS Pricing Modeler* is situated in the PaaS layer of the cloud stack, and its main functionality is to estimate the price per hour and charges of an application before deployment, as well as to calculate its charges after its operation.

The *Iaas Pricing Modeler* is situated in the IaaS layer of the cloud stack, and it is used to estimate the price per hour and the charges of a VM before deployment, as well as to calculate its charges after its operation. The goal of the IaaS Pricing Modeler is to provide energy-aware price estimation related to different IaaS level operations that may be envisioned in order to take the most energy-efficient course of action.

A variety of cost functions and pricing models can be implemented within the Pricing Modeler components. A *cost function* calculates the costs that a provider faces during its operation in order to offer his services. It is a mathematical formula associated with a certain action. The providers forecast their expenses associated with their services, to determine what pricing strategies to use in order to achieve the desired profit margins. A *pricing scheme* is another mathematical function that dictates the way of making revenues from the customers. For the determination of the right pricing strategy, the decision on the objectives of the strategy and good market knowledge are a necessity. The prices resulting from the model are used as part of the Service Level Agreement (SLA) that is contracted with the customer. Thus, they are needed both during the negotiation phase with the customer and during the billing phase.

The basic purpose of the Pricing Modeler components is to enable the use of appropriate cost and price functions for the provider of the corresponding cloud layer. The cost function of a PaaS provider differs from that of an IaaS one, which is straightforward. A PaaS provider neither pays the electricity bill, nor owns any physical machine. However, a PaaS provider could act as a broker providing cloud services from different IaaS providers. Thus, the costs of a PaaS provider are based on the contract that he has with the IaaS provider(s), and the licenses of the operation systems and administration operations. We envision that our proposed energy-aware pricing schemes (Sect. 4) and service plans (Sect. 5) make sense mainly for the IaaS providers, since there are direct cash flows between them and energy providers, but at the same time motivate the upper layers to become energy-efficient.

3 Adopted Pricing Schemes

Let us provide a brief overview of the current pricing schemes adopted within the cloud market. We studied twelve well-known worldwide providers that offer cloud services. Most of them are IaaS providers but many of them are also PaaS. The most common pricing scheme is the "*pay-as-you-go*" one, but monthly or yearly subscriptions can be found too. Furthermore, Amazon provides one different scheme; the "*spot instances*".

In the "pay-as-you-go" scheme the customer pays only for the resources that he uses. There is no minimum fee and the total price that the customer pays depends on the resources needed, as well as on the operating system used on top. The charge is made per hour, while usually 740 h correspond to one month. When the "pay-as-you-go" scheme is used, the customer can choose the amount of a variety of characteristics that will compose his VMs. The basic characteristics of the VMs are the capacity of the CPU and the memory. Depending on the service that the customer is running, more resources can be purchased. Such resources may be storage, data transfer and the operating system, depending on the provider. The pricing is done differently per resource. Usually, the *capacity* is charged per hour, while the *data transfer* and the *storage* per GB per month. Some companies also charge for each request, or per million I/O requests, or per HTTP request, or per GB of data processed. The latter scheme is used mostly when the applications running are short-term and their workloads are unpredictable or changing over time.

The other popular scheme used is the *periodic payment* (e.g., monthly, semester, yearly subscriptions, etc.) or *pre-payment*. The customers pay or pre-pay the use of specific resources, having a discount on the hourly charges. Usually under these schemes, if the needs of the customer change, the resources reserved for him cannot be returned and the amount is not refunded. But on the other hand, if the customer needs more resources, he can always purchase under the "pay-as-you-go" scheme.

It is worth noting that some providers offer an on-line cost calculator to their potential customers. Such tools allow on-the-fly addition of the type and amount of resources needed and the upper bound of the amount of money that the customer will pay at the end of the month under 100% utilization of the resources and the "pay-as-you-go" scheme (or the exact amount on the monthly one). The estimates for

the "pay-as-you-go" scheme are done based on 730–750 h per month. This scheme is used mostly for applications with more predictable usage patterns.

Most of the IaaS/PaaS providers nowadays use the "pay-as-you-go" scheme. *AT&T* is a representative example. The customer may choose the size of the processing capacity, the memory and the system storage that will compose his VM. For AT&T's PaaS, the customer selects a package based on his needs, builds his application and begins using it. The packages may include networking tools, email, web-based support and the option to add mobile users. The customer pays a per-device per-month fee [4]. *GoGrid* is an IaaS provider of computing, network and storage resources. It offers hourly, monthly, and annual cloud server pricing. Under the hourly "pay-as-you-go" option, there is no commitment, and the customer pays per hour for the resources used. The resources can be increased or decreased depending on the needs. Monthly or annual cloud server plans provide discounts in hourly charges, since the customer commits to a specific period of time of resource use [5]. In *Terremark* also the hourly price depends on the virtual processors (VPUs), the memory, the system storage configuration, and the operating system used [6]. *Microsoft* charges its VMs by the minute. In case of a monthly or yearly subscription the discount can fluctuate from 20 to 32%. Microsoft also offers a

Table 1. Comparison of cloud providers' pricing schemes.

Provider	Pricing scheme	Provided cloud services
Amazon [9]	On-Demand Instances, Reserved Instances, Spot Instances	Standard, Second Generation, Micro, High Memory, High CPU, Cluster Compute, High Memory Cluster, High I/O, High Storage, and according to the operating system
RackSpace [10]	Pay-as-you-go	Cloud Servers: Size, Disk, vCPUs, Public/Internal Network, operating system
GoGrid [5]	Hourly, monthly, semiannual, and annual cloud	RAM, Cores, Storage
Microsoft [7]	Pay-as-you-go, semester, year	CPU, RAM
Terremark [6]	Pay-as-you-go	Memory, VPU
AT&T [4]	Pay-as-you-go	Capacity, memory and system storage
Google [11]	Pay-as-you-go	Virtual Cores, Memory, Local disk
OpScource [8]	Pay-as-you-go, monthly	Size, Disk, vCPUs, Public/Internal Network, operating system
SoftLayer [12]	Pay-as-you-go, monthly	Core, RAM, storage, operating system
HP [13]	Pay-as-you-go	Core, RAM, storage, operating system
Engine Yard [14]	Pay-as-you-go	Core, RAM, storage
Acquia [15]	Pay-as-you-go	Core, RAM, storage

larger discount under the pre-paid monthly fee [7]. *Opsource* bills the customer only when the server is actually running [8]. For servers that are in a non-running state (stopped), the customer pays only for the storage that the server is using.

"On-demand instances" of *Amazon* correspond to the "pay-as-you-go" pricing scheme mentioned before. The customers pay for compute capacity by the hour with no long-term commitments. The notion behind the "reserved instances" is the reservation of the resources before their use for a specific amount of time. The customers can make a low, one-time payment for each instance that they reserve and in turn receive a significant discount on the hourly charge for that instance. Amazon provides three types of instances; for *light*, *medium*, and *heavy* utilization.

However, Amazon also provides the "*spot instances*" scheme. The customer buys the unused Amazon EC2 capacity and runs it until the price of the instances bought becomes higher than his bid. The *spot price* changes periodically based on supply and demand, and customers whose bids meet or exceed it, gain access to the available spot instances [9].

In the following table we present the providers examined, the employed pricing scheme, as well as the different VM features that the customer pays for (Table 1).

4 Energy-Aware Pricing

In this section, we propose novel pricing schemes for charging services based on their actual consumption to ensure energy efficiency of cloud resources. The energy models and the real-time monitoring mechanisms and measurements by ASCETiC make possible the creation of new energy-aware pricing schemes.

4.1 Why Energy-Based Pricing?

In Sect. 3, we observed that cloud IaaS providers mainly charge for their resources —which come in the form of VMs with specific performance characteristics— on the basis of fixed rates per unit of time. The rate levels depend on specific VM characteristics, such as CPU speed, memory, network bandwidth, etc. In certain cases, the pricing varies dynamically in time and depends on bids made by other IaaS customers. In any case, IaaS prices do not depend on energy usage —at least not explicitly, since IaaS providers strive to recover their factor (e.g., energy) costs through the appropriate selection of pricing levels—.

At the same time, applications take decisions which can have an important impact on both energy consumption and performance. An example of such a decision is the level of parallelism in the event of multiple tasks scheduled on many different VMs: the application has the choice of the parallel execution of a number of tasks on many different VMs instead of using only a few. Choosing a large number may prevent server consolidation from reaping all the potential energy gains. Actually, since pricing is not energy-dependent as discussed above, applications would go after the maximum level of parallelization possible, i.e., they will utilize all available VMs (or, at least it will not be in their interest not to do so). Thus, even though the great level of parallelism makes

an application to have unnecessary low latency, it may incur unnecessarily high energy costs (by requiring a large number of physical servers to host the VMs). These increased energy costs are carried over to increased IaaS prices and so lower profit levels for the IaaS providers.

We propose to use IaaS prices which dynamically depend on the energy usage of applications. Under such a scheme the applications will be aware of the economic impact of their decision and so they will have the incentive to take energy costs into account, e.g., when they decide on the level of parallelism. Applications will themselves trade energy for performance according to their preferences, and not let other entities such as IaaS providers do it instead (through server consolidation) on the basis of guesswork about their preferences. Indeed, task scheduling at the application level may be more energy and performance effective than server consolidation by the IaaS providers since it is the applications which know what should be run in parallel and what should not.

Another reason that makes energy usage based prices desirable is that it is common for energy prices to vary in time for various reasons (e.g., varying availability of energy sources, time-of-day pricing, demand-response schemes).

4.2 Energy-Aware Pricing Schemes

In Table 2, we define some useful notions to be used in what follows:

Table 2. Terminology for energy-aware pricing.

Term	Description
Price	The (time) average charge incurred by a VM (or an application) per unit of time measured in euros per hour
Charge	The total charges incurred by a VM (or an application) measured in euros
Energy price	The price per a unit of energy, in euros per Watt seconds
Static price	The portion of price not explicitly depending on energy consumption; usually it depends on the static characteristics of a VM (e.g., CPU speed, Memory, maximum network bandwidth, etc.). It could be also the result of a market mechanism, e.g., auction for computing resources
Static charge	The total charge due to static prices
Billing	The calculation of a price or charge incurred by a specific VM based on past usage
Prediction	The calculation of a price or charge estimate concerning the future usage of a specific VM, given a prediction of its energy (or power) consumption
Pricing scheme	A formula for computing the price

The pricing schemes we present below are based on the costs of an IaaS Provider during its operation or on predicted charges based on estimates of future usage. Such costs take into account the energy consumption of a VM.

4.2.1 Two-Part Tariff Pricing

The actual form of IaaS price could be comprised by two parts: a fixed one, a, depending only on static information of a VM, and a dynamic one, b which depends on the average power usage. As an example we have the following simple scheme: a is a fixed part based on static VM characteristics, and b is the average power usage multiplied with the price per watt-hour (Wh).

Thus, the price p of a VM (starting at time 0 and up to time T) is computed by the formula

$$p = \frac{1}{T}\int_0^T p_{static}(VM,t)dt + \frac{1}{T}\int_0^T p_{energy}(t)W(t)dt \tag{1}$$

where,

VM: a parameter identifying the characteristics of the VM
$p_{static}(VM,t)$: the static price of VM at time t
$p_{energy}(t)$: the energy price at time t
$W(t)$: the power usage of the VM at time t.

We assume that the energy price changes only at the time instants $T_0 = 0 < T_1 < T_2 < \ldots$, and let the energy consumption during the corresponding time period be as given by the red curve in Fig. 2.

Then the total charges $C(T) = \int_0^T p_{energy}(t)W(t)dt$ incurred up to time T can be calculated from $C(T_k)$ as

$$C(T) = C(T_k) + p_{energy}(T_k)\int_{T_k}^T W(t)dt \tag{2}$$

Thus, in order to be able to calculate the charge for any VM one must keep track of $C(T_k)$, i.e., the charges incurred up to the last price change, the current energy price $p_{energy}(T_k)$ and the energy $\int_0^{T_k} W(t)dt$ consumed (by this VM) up to the last price change. Then the energy consumption $\int_{T_k}^T W(t)dt$ appearing in (1) can be computed as the difference $\int_0^T W(t)dt - \int_0^{T_k} W(t)dt$. Hence, on a price change one must iterate through all the VMs in the infrastructure and update $C(T_k)$, $\int_0^{T_k} W(t)dt$. The energy price p is computed from the total charge $C(T)$ as $p = C(T)/T$.

The term $\frac{1}{T}\int_0^T p_{static}(VM,t)dt$ represents the static price of the VM based on its own characteristics. If the static price does not vary in time, i.e., $p(VM,t)$ is constant in the time parameter t then no time averaging is necessary. If it does vary then similarly to the above analysis, the total static charge $\int_0^T p(VM,t)dt$ up to time T can be written as $\int_0^{T_k} p(VM,t)dt + p(VM,T_k)(T-T_k)$, i.e., the total static charges incurred up to time T_k plus the static charges from that point onwards. Thus, in order to keep track of the static charges incurred by any VM, the total static charge up to the last static price change[1] should be stored (for each VM). Consequently, every time the static prices changes one must update the static charges for each VM in the infrastructure. The static price up to time T is computed from the static charges as $\int_0^T p(VM,t)dt/T$.

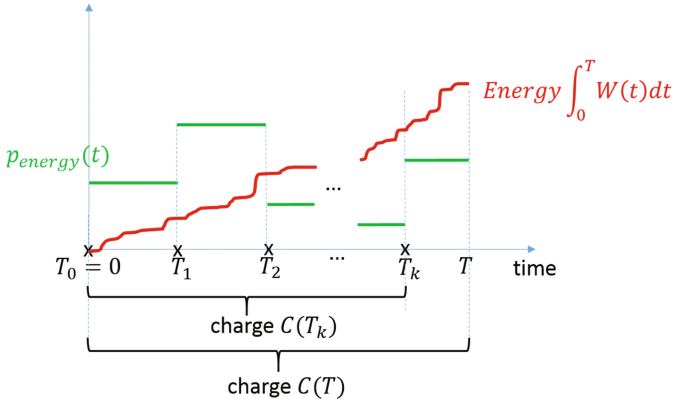

Fig. 2. Recursive calculation of energy charges $\boldsymbol{C(T)}$ up to time $\boldsymbol{T}$ by the energy charges $\boldsymbol{C(T_k)}$ and the energy charge during the time period from $\boldsymbol{T_k}$ up to $\boldsymbol{T}$, where $\boldsymbol{T_k}$ is the last instant the energy price changed prior to $\boldsymbol{T}$ (Color figure online).

4.2.2 Two-Part Tariff with Energy Saving Discounts

A disadvantage of the dynamic usage price presented in Sect. 4.2.1 is that the actual energy that an application may use is not known by the developers at the time the SLA is established. A simple alternative is to pay a lump sum and then apply a discount based on the actual power consumption. Hence, we could use the following two-part price: a is a fixed price based on static info of a VM which also incorporates energy costs through the historical average power consumption, and b is a price discount depending on the level of power savings below the historical average. In this way it is not possible to pay more than the lump sum initial payment.

[1] For example, if the static price is the spot price of a market mechanism.

More specifically, the price p is computed by the formula

$$p = \frac{1}{T}\int_0^T p_{static}(VM,t)dt + \min\left\{\frac{1}{T}\int_0^T p_{energy}(t)W(t)dt - \frac{1}{T}\int_0^T p_{energy}(t)W_{nominal}dt, 0\right\} \quad (3)$$

where, $W_{nominal}$: the nominal average power consumption, i.e., the power consumption already accounted for in the static price. Any average power consumption above $W_{nominal}$ does not increase price above the (time average) static price. Deviations below $W_{nominal}$ result into a proportional discount.

4.3 Linearly Increasing Energy-Based Pricing

In both aforementioned pricing schemes, we assumed that the price of energy could potentially vary in each epoch. However, such schemes do not consider any direct relation between the energy price and the total energy consumption. Let us consider an energy provider facing energy consumption bursts (e.g., during summer) that he reasonably would like to avoid. A traditional pricing scheme adopted by the majority of the energy providers, is to provide a lower price per energy unit during the less bursty periods (e.g., day/night).

Motivated by this approach, we investigate how an IaaS provider could provide the necessary incentives to his customers in order to shift their energy demand to less bursty periods. In this scheme, we assume the price per energy unit based on the total consumed energy to be a linear increasing function.

It should be mentioned here that other approaches (e.g., exponential function) may also be applied, in order to capture the notion of setting a higher price per energy unit, as more energy is consumed during an epoch. The slope of the charging function will be set by the IaaS provider based on the factors affecting his own cost function (e.g., charging scheme or/and SLAs between IaaS and energy provider).

For the linear assumption, p_{energy} can be written as $cW(t)$, assuming that c is a constant parameter set by the IaaS provider, showing how aggressively p_{energy} will increase with respect to the total energy consumption. In order to prevent IaaS provider to charge arbitrarily high prices, we set an upper bound, such that $cW(t) \leq p_{energy\_upper}$.

Thus, the price p is computed by the formula

$$p = \frac{1}{T}\int_0^T p_{static}(VM,t)dt + \min\left\{\frac{1}{T}\int_0^T cW^2(t)dt, \frac{1}{T}\int_0^T p_{energy\_upper}(t)W(t)dt\right\} \quad (4)$$

4.4 95th Percentile Rule

The 95th percentile rule is a widely used pricing scheme in telecommunications for charging the transit traffic sent by lower-tier ISPs. By employing this scheme, transit ISPs intend to penalize lower-tier ISPs in case of traffic bursts.

A similar pricing scheme could be employed by IaaS providers for penalizing bursts of the consumed energy. To implement this scheme, it is assumed that the energy consumption within the infrastructure of an IaaS provider is measured or sampled and recorded (e.g., log file)[2]. At the end of each billing cycle (e.g., every month), the energy consumption samples are sorted from highest to lowest, and the top 5% of data is thrown away. The next highest measurement is the 95th%, and the customer will be billed based on that energy consumption.

We let $l^*(t)$ denote the 95th% measurement of the energy consumed by the customer at time t. Then, l^* is defined as $max\{l|P(W > l) \geq 0.05\}$.

Thus, the price p is computed by the formula

$$p = \frac{1}{T}\int_0^T p_{static}(VM, t)dt + \frac{1}{T}\int_0^T p_{energy}(t)l^*(t)dt \tag{5}$$

5 Service Plans

In this section, we introduce service plans for the IaaS/PaaS provider to facilitate the evaluation of opportunity costs by offering multiple mutually exclusive service plans to its customers. The plans are intended to form the basis of business level contracts between layers by clearly specifying the responsibilities between the participating layers. Service plans is just one example of a method which IaaS/PaaS providers may use to facilitate energy-aware adoption, and others may exist. They are given here to complement the discussion on ASCETiC adoption implied by the energy-based pricing schemes proposed in Sect. 4.

The proposed service plans are intended to form the basis of business level contracts between cloud layers by clearly specifying the responsibilities between the participating layers. These responsibilities are the combination of

a. an SLA agreement,
b. a pricing scheme, and
c. a clear understanding of adaptation semantics, i.e., a specification of the originator and the means of corrective actions on the event of an SLA violation or just before it happens.

It is important to note that the plans need not be part of the ASCETiC architecture as such, as they are defined by combinations of instances of (a, b, c) above.

[2] For example, in the ASCETiC framework, the *IaaS Energy Modeler* component is the principle component for predicting energy usage and generating historic logs of usage [3].

On the other hand, the service plans are essential in evaluating opportunity costs and therefore affect business and technological decisions.

A key idea is that multiple plans should be offered at the same time, with multiple choices existing along two axes: the degree of adoption of energy-aware functionality, and the level of service performance.

Adoption axis: A successful adoption strategy should include plans similar to the ones offered in legacy architectures in order to not exclude customers (i.e., layers, applications) which

i. do not want to change method they are charged with to an energy-based one, and/or
ii. are not capable of exploiting the additional features of an energy-aware architecture.

Non-legacy plans which take information on energy into account, i.e., when either constituent of (a, b, c) involves energy-related terms, should be offered alongside with legacy ones. This coexistence will allow the gradual adoption of energy-aware elements of the architecture. The menu of plans offered should accommodate all customer types in respect to their adoption degree. For example, we could identify three degrees of adoption:

(1) legacy customers, i.e., described by (i, ii) above,
(2) non-legacy customers which prefer legacy payments, i.e., described by (i) but not (ii), and
(3) non-legacy customers which do not belong to either (i) or (ii), i.e., they have fully adopted the architecture and energy-aware payment models.

Performance axis: The second axis along which customers are categorized is performance. If multiple performance measures exist, such as response delay and reliability then the second axis is actually multidimensional. Higher performance means lower latency and/or higher throughput and is normally associated with costlier plans, while low performance is a budget option.

Figure 3 depicts these two axes along with five illustrative service plans which intend to cover most of the space defined by the axes:

HiPerf: geared towards high performance legacy customers. Normally it is associated with SLAs with performance-related terms. As these customers are not energy aware, the lower layer is solely responsible for adaptation actions.

Budget: a budget plan for legacy customers, which usually comes with loose or no performance guarantees specified in the SLA. Again the lower layer is solely responsible for adaptation.

Vegan: a budget plan for type (2) customers, i.e., non-legacy customers with legacy payment plan. SLA terms impose strict limits on maximum power usage (within a specific time window). The lower layer is responsible for ensuring these limits are never violated. Other than that, the customer is free to adapt in order to make best use

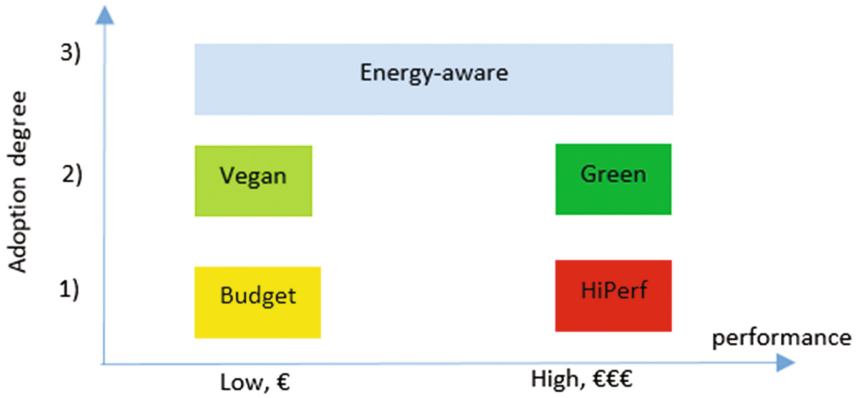

Fig. 3. Adoption degree of service plans with respect to the charging method and performance.

of system resources without exceeding the power limits. As the payment plan is legacy (i.e., not depending on energy consumption) the customer has the incentive of doing the most out of its energy-aware capabilities for the amount of price it is paying.

Green: a more high-end plan than Vegan, for type (2) customers. The share of responsibilities is the same as Vegan the only differences being that the power limits are higher at a higher price. Because the power limits are higher, this plan can be used by customer seeking higher performance.

Energy-Aware: a plan for non-legacy customers with energy aware payments. Since energy consumption is charged by the lower layer, it is the sole responsibility of the customer to make good use of energy-efficient actions. SLA terms could prevent overcharging due to excessive energy consumption, by either limiting the maximum power usage or maximum energy-related charge. (In the case the latter is exceeded the provision of service by the lower layer could temporarily be suspended until energy charges drop below the limit.) Both high and low performance can be accommodated under this plan, as the performance/cost tradeoff is determined by the customer's adapting actions.

These plans are defined such that they span most of the space defined by the two axes. This is to ensure that as many customers as possible are accommodated within a system employing this architecture. The existence of intermediate adoption degrees (customer type (2)) also allows gradual adoption in smaller steps, instead of a single big one. This reduces the adoption costs at each step and makes full adoption more likely.

Finally, the pricing strategy, i.e., the selection of prices in the pricing schemes, should be such that services higher in the adoption degree axis are more competitive than lower ones. In other words, the pricing of service plans should provide incentives for adoption of energy-aware technologies by the customer. Thus the service plan price differences could offset part of the adoption costs and act as subsidies by the IaaS/PaaS provider. Of course pricing should also consider competition with legacy clouds.

As an example of an adoption path towards full energy-awareness, consider an application which does not require small response times and desires a low cost service.

Initially it subscribes to the Budget service plan as the closest match. Since low cost is important for the application, it has the choice of subscribing to the "Vegan" plan at a smaller but still fixed price. If the application does not employ any energy-aware capabilities through e.g., energy-aware scheduling of requests on VMs, then soon the tight bounds on power consumption of the "Vegan" plan will be violated and so performance will degrade. The only way of avoiding this to happen is for the application to develop energy awareness. Of course this involves extra "ASCETiC adoption" costs as explained before, but the lower "Vegan" price should offset these.

Once the application develops energy-awareness, it could evaluate if the "Energy-aware" plan is better. It may be a better solution than "Vegan" since there will be times, e.g., cloud-wide congestion, which significantly degrades performance. In these situations, through the "Energy-aware" plan, the application can avoid big performance drops by being able to use more resources at will at a temporarily higher cost.

6 Conclusions and Future Work

The ASCETiC project aims to provide novel methods and tools to support software developers aiming to optimize energy efficiency at the different layers of the cloud stack. The Pricing Modeler is a component within the ASCETiC architecture, which is responsible for providing energy-aware cost estimation related to the operation of applications, as well as billing information.

From our market analysis on cloud service pricing, we observed that cloud providers mainly charge for their resources on the basis of fixed rates per unit of time, without taking explicitly into account the energy usage. In response, we proposed four novel energy-aware pricing schemes to enhance IaaS providers choosing their optimal pricing strategy, reflecting also our target for incentivizing the customers to be energy-efficient. The proposed pricing schemes differ in terms of aggressiveness with respect to the charging of energy consumption bursts. Furthermore, we presented a set of envisaged service plans intending to facilitate the gradual adoption of the ASCETiC architecture.

Our future work will focus on proposing new energy-aware pricing schemes, as well as evaluating them based on different scenarios. For example, each pricing scheme could be selected by a cloud operator based on the type of the applications running within its infrastructure. Another dimension of our research is to investigate and evaluate scenarios assuming competition among cloud providers employing different pricing schemes, as well as consider the equilibriums (if any) in the cloud market.

Acknowledgements. This work is partly supported by the European Commission under FP7-ICT-2013.1.2 contract 610874 - Adapting Service lifeCycle towards EfficienT Clouds (ASCETiC) project.

References

1. ASCETiC, EU FP-7 project. http://ascetic-project.eu/
2. Mell, P., Grance, T.: The NIST definition of cloud computing. Nat. Inst. Stand. Technol. **53**(6), 50 (2009)
3. ASCETiC Deliverable D2.2.3 Architecture Specification – Version 3, Public Deliverable, December 2015
4. AT&T Cloud Services website. http://www.business.att.com/enterprise/Portfolio/cloud
5. GoGrid website. http://www.gogrid.com
6. Terramark website. http://vcloudexpress.terremark.com
7. Microsoft Windows Azure website. http://www.windowsazure.com/en-us/pricing/calculator
8. Opsource website. http://www.opsource.net
9. Amazon Elastic Compute Cloud website. http://aws.amazon.com
10. Rackspace website. http://www.rackspace.com/cloud
11. Google Cloud Pricing website. https://developers.google.com/storage/pricing
12. Softlayer website. http://www.softlayer.com
13. HP cloud website. http://www.hpcloud.com/pricing
14. Engine Yard website. https://www.engineyard.com/products/cloud
15. Acquia website. http://www.acquia.com/cloud-pricing