Energy-based Cost Model of Containers Provisioning on Clouds

Abstract Cloud computing revolutionized the development and execution of distributed applications by providing on-demand access to virtual resources. Containerization simplifies management and support of the cloud infrastructure and applications. Clouds typically are consumed in a pay-as-you-go pricing model. However, when applied to containerized environments, such traditional models do not consider resource utilization values, leading to inaccurate estimates. Moreover, these models do not consider energy consumption, a dominant component of the data center's total cost of ownership. This paper proposes Energy Price Cloud Containers (EPCC), a cost model based on energy consumption that accounts for containers' effective resource utilization. We compare EPCC with AWS Fargate to highlight the benefits of using an energy-based pricing model. Thus, by comparing the cost of an application running using Amazon Web Services (AWS) Fargate with the estimated cost of that application in Energy Price Cloud Containers (EPCC), it is possible to identify the benefits of using an energy-based pricing model. The weekly costs estimated when running computational resources at EPCC vary between US$ 2.31 and US$ 10.59. In contrast, when estimating the same amount of resources on AWS Fargate, the costs vary between US$ 2.71 and US$ 29.94. EPCC resulted in a cost reduction of up to 35%.

Keywords: Pricing Model; Containers; Cloud Computing; Energy Consumption.

1 Introduction

Virtualization technologies are in constant evolution. Containerization, based on OS-level virtualization [1, 2], provides scalability and flexibility to develop and deploy applications while simplifying management and adapting to customers' needs. Cloud services leverage virtualization technologies at scale. Business models are based on dynamic on-demand resource allocation, pay-as-you-go pricing, and resource consumption [3]. Several cloud providers offer elastic virtualization services, e.g., Amazon AWS, Microsoft Azure, Google Cloud, Rackspace, Heroku [4]. An increasing number of organizations rely on cloud computing as their core pool of resources to serve their own customers. Cloud-related expenses represent a considerable part of the Information Technology (IT) organization budget [5]. A precise specification of resources to be allocated and provisioned is crucial to improve performance and costs while addressing each service's complexities [6, 7].

Infrastructure as a Service (IaaS) providers adopt various pricing models based on monthly contracts, licenses, and Service Level Agreement (SLA) policies. For example, Amazon, Microsoft Azure, and Google Cloud Platform offer pay-as-you-go services charging a predetermined price for each resource, per task, or by the number of calls of a service [7]. From the cloud provider perspective, data center (DC) energy consumption stands out among all operational costs [8]. This directly impacts the costs seen by customers in the cloud service catalog. Virtualization contributes to efficient energy management through resource consolidation and isolation techniques, improving DC resource utilization [9]. Compared to traditional hypervisor-based virtualization, containerized infrastructures demonstrate even higher efficiency by sharing the available physical resources and the operating system (OS) kernel.

Virtualization technologies and application settings influence the utilization level of computational resources such as CPU, memory, networking, disks, etc. For instance, bandwidth and latency requirements may come from different network settings used by an application (*i.e.,* NAT, bridge, host-only [8, 10]) as well as from the properties of the workload. Consequently, these aspects also have a direct impact on energy consumption. Cloud customers typically face a trade-off between reducing resource consumption (*i.e.,* cost) and improving performance. From the customer perspective, reducing the energy consumption is not a priority since there is no transparency about how energy resources are consumed or any financial reward [2, 11, 12].

We proposes Energy Price Cloud Containers (EPCC). EPCC brings energy efficiency into the equation by defining an adaptable pricing model that accounts for energy consumption as a resource utilization function, not just fixed time-based and allocation-based quotas. This provides the means to establish a clear financial incentive to save energy since it directly ties energy consumption to service costs. Our assessment of the impact of container utilization on energy consumption reveals that each resource has a different impact on the final estimates. Energy consumption rapidly increases proportionally to the percentage of CPU utilization. In contrast, memory and network observe a milder increase. Storage resources typically display stable energy consumption, even for I/O-intensive applications. This fine-grained analysis led to the stratification of containers' energy consumption. EPCC formalizes these components in a precise and extensible pricing model.

The contributions of this work are threefold: (i) an energy-aware cost model to containers; (ii) stratification of container energy consumption; and (iii) a comparison of energy consumption among servers, virtual machines, and container environments.

We compare estimated energy costs from Amazon Web Services (AWS) Fargate with EPCC cost model. AWS Fargate follows a pay-as-you-go charging model that does not consider the resource utilization levels of leased containers. Since the AWS Fargate invoice comprises all business costs, we consider that AWS Fargate energy costs represent 5%, 10%, and 15% of the total business costs [11].

EPCC estimated weekly values vary between US$ 2.31 and US$ 10.59, while AWS Fargate US$ 2.71 and US$ 29.94, a reduction of up to 35.39%. The AWS Fargate account was only cheaper than our EPCC model when we estimated that the cost of energy in the AWS Fargate model represents 5% of the total value of the contracted service. However, when the container is allocated and remains for a long period without physical resource demand, the server usage is set to idle. The entire allocation period is counted with the total contract for AWS Fargate, even the server usage idle period. Thus, a container application is hampered by the AWS Fargate cost model if the allocation resource is overestimated.

This paper is organized as follows. Section 2 addresses background and prior work on cost models for virtualized resources. Section 3 describes and applies the method to provide a breakdown of energy consumption based on container application resource utilization. Section 4 presents EPCC. Section 5 presents a case study to exercise EPCC using the AWS Fargate pricing table. Section 6 concludes and proposes future work.

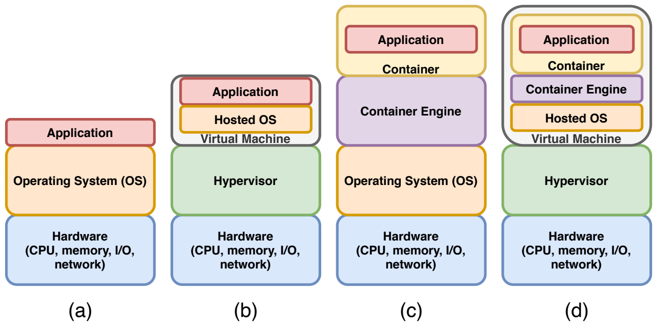


Figure 1 Common approaches for providing computational resources to an application.

2 Cost models for virtualized resources

The consolidation of computational resources directly influences the DCs energy reduction. Organizations have been migrating their services to the cloud computing service model since its emergence. Organizations still might maintain their DCs to ensure strict control of management policies or because of legacy software. Although the concentration of physical resources on large DCs reduces global energy consumption [13], they still are an energy bottleneck.

In 2015 the energy consumption reached 416 TWh with a forecast to double every four years [13]. In the United States the DCs consumed more than 90 TWh [8]. The required energy supply system causes an economic and environmental impact. Thus, green clouds, renewable energy supplies, and cost models have become interesting topics for scientific research. [14, 15, 16, 17, 18, 19].

Energy cost represents about 50% of the DC operating expenses [20, 9]. Providers constantly look for new methods, equipment, and practices to improve their DC energy efficiency. In contrast, cloud tenants typically are not aware of the energy consumed by their applications, nor are they rewarded for improving it.

2.1 DC Servers Virtualization

Virtualization allows servers to be consolidated, therefore improving energy usage and simplifying DC management [21]. Although it exists since the mid-1960s, virtualization has seen several improvements specially after its adoption by cloud infrastructures to transparently provide resources such as CPU processing, storage, and virtual environments. This allows IaaS providers to grant access to remote, configurable, and shared sets of computing resources. IaaS providers can efficiently afford and make resources available to their tenants while minimizing management, communication, and energy costs [22, 23].

There are two primary technologies used to implement virtualization: (i) *hypervisor-based*, which requires each Virtual Machine (VM) to load a complete OS (leading to higher overhead); and (ii) *container-based*, a lightweight approach that shares the host OS kernel among all instances [2].

Containers provide an abstraction at the application layer by packaging the code and the necessary dependencies for its operation [24]. Figure 1 presents some of these virtualization environments adopted by organizations, including bare metal (a), VM Type-1 (b), container on bare metal (c), and container atop VM (d). Using containers does not restrict the use of VMs; in fact, it allows the encapsulation of applications in several layers [25]. Docker is one of the most prominent containerization technologies. Docker started in 2016, reaching U$761 million in market value, and this market could grow more than 35 times up to 2020 which would represent U$27 billion [26]. Gartner [27] expects that up to 15% of enterprise applications will run in a container environment by 2024, up from less than 5% in 2020, hampered by application backlog, technical debt, and budget constraints.

The transition from internally hosted services to cloud computing can result in 4-5x improvement in resource utilization efficiency [7] – up to 6x with container-based virtualization. The adoption of virtualization is, therefore, a solid technological trend – but what about energy? How to factor in economic aspects such as supply and demand while encouraging tenants to use DC resources consciously?

2.2 Cost Models for VMs

The main pricing models used by cloud providers consider value, supply, and market. The value aspect focuses on customer demand, while the supply aspect focuses on cost. The market-based model seeks a balance between supply and demand. Cloud providers offer a vast range of services comprising infrastructure models and ways to allocate, use, and reserve resources. This evolution was naturally followed by creating cost models suitable to an extensive range of tenants with different profiles [7]. However, despite constant improvements to products and services, current cost models still have gaps.

Figure 2 presents the evolution of AWS cost models. The X-axis represents the number of sales for each service model, q. The Y-axis represents the service pricing per unit, p. Over the years, AWS has designed new service and cost models focusing on the market and operating costs to fill provisioning gaps. We identify gaps in cloud computing in two distinct moments: (i) in its early days (2009), when AWS concentrated its resources on centralized DCs infrastructures; and (ii) in current days (2019), when AWS offers geographically distributed and fully decentralized infrastructures. Supply and demand was the foundation of the core services, providing discounts up to 90% compared to the same server in the AWS standard offer model. In 2015 AWS started offering a specific model for executing fault-tolerant workloads. In the same year, AWS products portfolio included two other pricing models for the spot service: *spot blocks*, in which instances have a specified duration; and *spot fleets*, sets of spot instances that run based on specified criteria. This flexibility allowed tenants to specify their resource needs with precision, reducing the resources' underutilization [7].

Figure 2 reveals that AWS had notable success in reducing underprovisioning – white areas in the diagrams. Competition from other providers such as Microsoft and Google led to creating of new and more efficient offerings. The existence of low-cost instances opened a new market niche for customers with relaxed requirements. From the providers' viewpoint, this increased the DC utilization rate by using of idle resources.

Although the evolution in service offerings is evident, there still are gaps to be addressed. This evolution is guided by providers' interests, policies, and perceptions of cost and service packagemodels. The information from tenants usually is disregarded, e.g., the optimization of their applications for lower energy consumption or politically correct actions, such as environmental preservation. Rewarding tenants for being aware of their energy consumption can be a strong motivator for optimizating applications. This paper highlights the need to fill the price gap per reservation, described in Figure 2 by the green hatched area in the *Present* years diagram.

2.3 Related Work

There are several researches in the specialized literature regarding energy consumption in computational environments. These studies examine different environments (*e.g.,* bare metal servers, VMs, containers), manipulate various resources (*e.g.,* CPU, network, memory, storage) in addition to having several purposes (*e.g.,* monitoring tool, predictions). Table 1 summarizes the related work. In short, the proposals have different characteristics, but none of them meet all the resources addressed by EPCC.

The proposals [11] and [12] introduced pricing models for IaaS clouds focusing on VMs and guided by energy consumption. Proportional-Shared Virtual Energy (PSVE) bases its pricing model on identifying the VM individualized energy consumption through the hypervisor, accounting the resources individually and collectively. Collective costs are prorated by VM according to the number of CPUs allocated in the period [11]. This benefits the customer's lowest cost and the providers' Property Total Cost (PTC), and goes to green IT actions. However, PSVE does not consider the relevance of containers in the computational clouds and restricts the proposed model only to VMs. Energy-proportional Profiling and Accounting in Virtualized Environments (EPAVE) [12] is a pricing model for accounting VM dynamic consumption and the proportional static cost of the cloud infrastructure, assigning general energy costs per VM. EPAVE and PSVE differ in the energy costs division policy and do not consider the network traffic energy consumption. A set of techniques to measure energy consumption was described by [28] and [29]. Regarding the workload impact on energy consumption, [29] claims that the impact on energy efficiency varies according to CPU models.

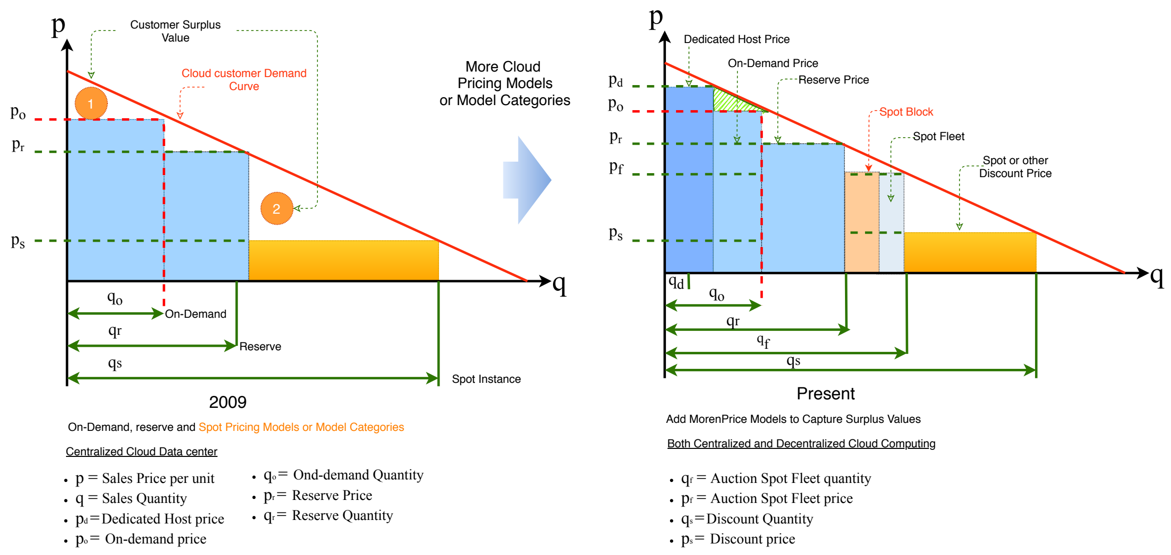

p
Customer Surplus Value
Cloud customer Demand Curve
1
2
po
pr
ps
q
qo
On-Demand
qr
Reserve
qs
Spot Instance
2009
On-Demand, reserve and Spot Pricing Models or Model Categories
Centralized Cloud Data center
• p = Sales Price per unit
• q = Sales Quantity
• pd =Dedicated Host price
• po = On-demand price
• qo = Ond-demand Quantity
• pr = Reserve Price
• qr = Reserve Quantity
More Cloud Pricing Models or Model Categories
Dedicated Host Price
On-Demand Price
Reserve Price
Spot Block
Spot Fleet
Spot or other Discount Price
pd
pf
qd
qf
Present
Add MorenPrice Models to Capture Surplus Values
Both Centralized and Decentralized Cloud Computing
• qf = Auction Spot Fleet quantity
• pf = Auction Spot Fleet price
• qs =Discount Quantity
• ps = Discount price

Figure 2 AWS Costs Models Evolution. Adapted from [7].

Autor	Objective	Environment	Resources	Objective
[11]	Analysis of energy costs in VMs and hypervisors	IaaS	CPU, network traffic	Pricing
[12]	Proportional energy cost model in environments based on cloud providers	IaaS	CPU	Pricing
[28]	Energy consumption measurement tool	DC	CPU, network, memory, and storage	Monitoring
[29]	Relates the energy efficiency of workloads to CPU consumption	Cloud providers	CPU and memory	Monitoring
[30]	DEEP-mon hardware energy monitoring of containers	OS	CPU, network, and memory	Monitoring
[31]	Container deployment cost model	IaaS and PaaS	CPU, memory, and storage	Pricing
EPCC	Container cost model based on energy consumption	IaaS	CPU, network, memory, and storage	Pricing

Table 1 Models and tools from the perspective of energy consumption of VM in DC or IaaS cloud providers.

Finally, two works stand out associated with virtualization-based on OS and containerized applications: (i) the DEEP-mon tool which monitors the CPU, network, and memory resources at the level of OS (containers) to measure energy consumption [30]; and (ii) a pricing model for deploying cloud applications based on microservices [31]. A research carried out in 2019 shows that more than 34% of companies that invest over $US\$100$ thousand in technologies are adhering to the container environments atop bare metal servers, migrating the applications of VMs for containers [32]. These companies aim to increase their performance, reduce infrastructure complexity, improve flexibility, and reduce costs. Containers are efficient, allowing operators to do more with less and manage critical applications with high availability. The usage of containers by companies ranges from modernizing legacy applications to analyzing big data [32]. Such research highlights the growing impact of containers for organizations, attesting the scientific gap in the containers pricing models study for cloud providers guided by energy consumption (one of the motivations of this work).

Cloud providers have been improving their service offered over the years, filling the gaps of underutilized processes and seek to maximize the energy efficiency in their DC [7]. However, on the client side, the providers'efforts fall short in the cost models offered. The cost models offered by providers do not provide economic benefits to their tenants to develop energy-conscious applications. A better cost model would benefit not only the customer's final cost but also environmental preservation. This action reflects the reduction of PTC of providers making the offer convenient for the provider and their tenants. The DCs must offer cost models that consider the energy consumption in their pricing, as proposed by PSVE [11]. However, this model only includes hypervisor-based environments, excluding containers in bare metal and therefore having lower support for containers per physical server in VM environments [32]. Moreover, providers offer container-based services priced by resource allocation, not utilization. This pricing and provisioning gap requires an appropriate cost model for containers focused on energy consumption so that tenants are encouraged to develop energy efficient applications. State of art in energy-aware consumption reinforces the importance of a pricing model for the IaaS context and energy monitoring methods for containerized applications. The focus of this work is to identify and address this gap regarding the energy-aware containerization-based pricing model.

3 Energy Consumption Analysis

This section presents an analysis of the energy consumption of computational resources. The objective is to establish a baseline to quantify and compare each resource's energy consumption impact (CPU, memory, storage, and network) in bare metal, VM, and container-based setups.

3.1 Experimental Protocol

We elaborated the experimental environment using the Grid5000 platform [33] Grenoble site (`https://www.grid5000.fr/w/Grenoble:Hardware`). The selected cluster, **Yeti**, comprises four Dell PowerEdge R940 computing nodes with Intel Xeon Gold 6130 (Skylake, 2.10GHz, 4 CPUs/node, 16 cores/CPU), 768Gb RAM, and an Intel Ethernet Controller X710 for 10GBE SFP+. A set of non-intrusive wattmeters monitors Yeti's energy consumption. The software comprises three layers: (i) Debian 10 GNU/Linux bare-metal installation; (ii) KVM hypervisor 3.1.0; and (iii) Docker 19.03.

We used four benchmarks to analyze energy consumption. Each benchmark applies a different workload to a specific resource. **StressNG** applies a configurable workload to the server's CPU [34]. **Stream** performs vector calculations to saturate the server's memory and CPU. The user defines the amount of memory consumed, CPU load, and a number of processes to be allocated [35]. **Fio** (flexible I/O) executes asynchronous read/write storage operations [36]. Finally, **iperf3** generates network traffic to stress bandwidth usage, allowing parallel communication flows between server and client [37]. These benchmarks allow us to observe each resource's behavior and its correlation to energy consumption [38].

We quantify the energy consumption of each physical resource for each active container. Our baseline is the bare metal setup (Figure 3 (a)). This baseline is compared to three other scenarios: (i) VM, (ii) Docker container, and (iii) Docker container inside VM, as shown in Figure 3 (b)-(d). Each benchmark was executed with the same workload 10 times.

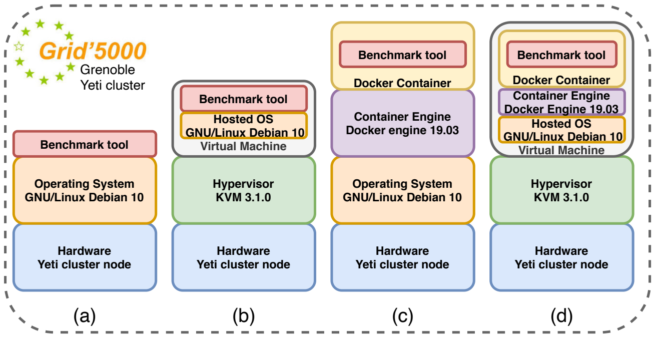


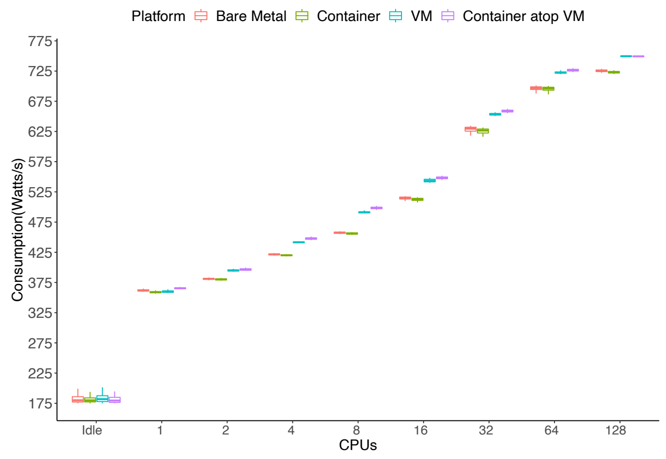
Figure 3 Experimental environment.

3.2 CPU Analysis

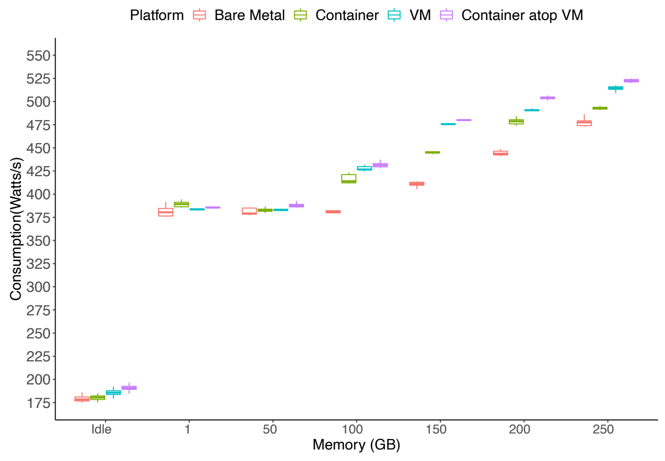
Figure 4(a) illustrates the energy consumption behavior according to the increase in CPU usage. The X-axis represents the number of threads (CPUs) assigned to the benchmark. The Y-axis represents the energy consumed (in joules) during the experiment. The idle consumption is around 175 J (1 J standard deviation) for all platforms. For each platform, we increased the number of benchmark threads from 1 to 128. Running one thread doubles energy consumption compared to idle. For 128 threads, energy consumption increases by more than 4x. Also, VM and container atop VM platforms consume more energy than bare metal and containers-only. Container atop VM consumption is slightly higher than VM only, except when the server is saturated. The energy consumption gap among platforms widens as CPU usage increases, mainly due to the management performed by the KVM hypervisor. Using only 1 CPU, the difference in consumption between environments is less 2%. For 8 CPUs, the difference is higher than 9%. This reveals that CPU usage is the essential component for a cost model; two customers may pay the same price for a service, but one consumes 3x more energy than the other.

3.3 Memory Analysis

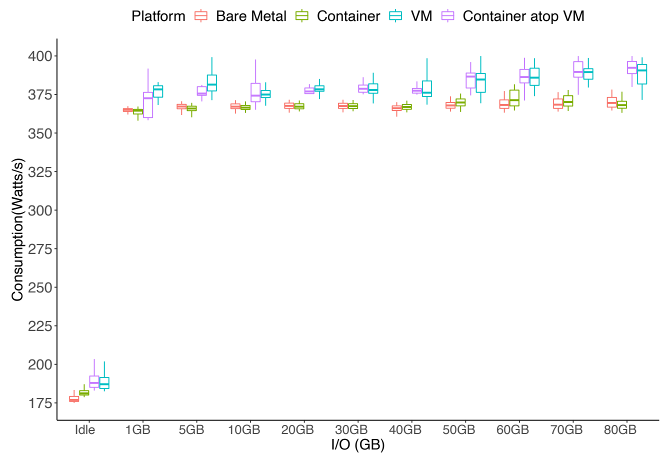
Figure 4(b) shows the results for the experiments with memory utilization. X-axis represents memory utilization in GB. Y-axis indicates energy consumption in joules. The experiments with STREAM [35] use seven memory workloads processed by only one thread. We compare the baseline (idle system) to six memory utilization values: 1 GB, 50 GB, 100 GB, 150 GB, 200 GB, and 250 GB. The energy consumption curve is visually similar to CPU but with a narrower range, up to 525 J. The experiments with 1 GB show an energy consumption increase between 375 J and 400 J. Increasing memory utilization to 50 GB, however, does not drastically change energy consumption. Also, subsequent experiments increase energy consumption by 7% on average compared



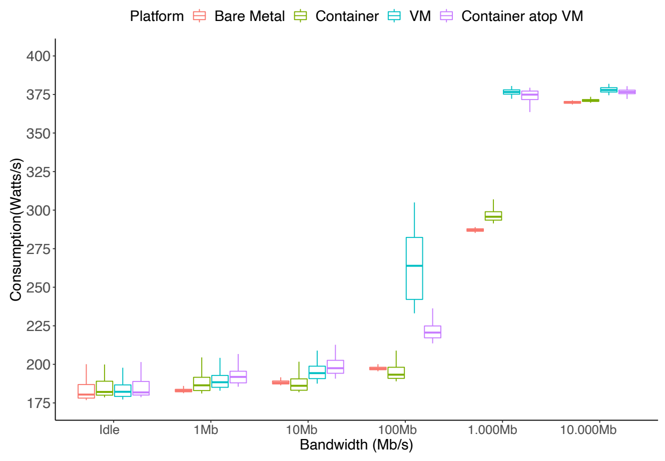
(a)



(b)



(c)



(d)

Figure 4 Physical resource energy consumption.

to the previous one. This reveals that, although memory utilization does influence energy consumption, its impact is not as pronounced as CPU.

Memory management represents an important resource demand for all platforms. The energy consumption gap compared to bare metal is more accentuated than the one for CPU resources. Not even the container setup can compete with bare metal. In the worst case, when comparing idle and 250 GB memory setup used for the container atop VM scenario, the energy consumption increases by 214%.

3.4 Storage Analysis

Consumption on SSD drives tends to be much lower, but many DCs still have hard drives. Thus, the storage analysis focuses on energy consumption from hard disks. Flexible I/O [36] is the storage benchmark selected for the evaluation. The storage demand varies from 1 GB to 80 GB. In all cases, the benchmark uses only one thread.

Figure 4(c) depicts the behavior of energy consumption with storage. X-axis represents the storage demand (in GB). Y-axis shows energy consumption in joules. Energy consumption from storage remains stable from 1 GB to 80 GB for all scenarios. The variation coefficient is around 8.4%. This reveals that increasing the storage demand does not impact energy consumption. Compared to idle, the energy consumption increases 2x.

3.5 Network Analysis

The last resource analyzed is the network. Network management in virtualized environments behave differently in terms of energy consumption depending on network type (e.g., host, bridge, overlay, macvlan) [39]. In our experiments, all scenarios were set up using bridges. We executed iperf3 [37] to create network workloads while evaluating energy consumption. We used two nodes: a client and a server. Energy consumption was measured on the server-side. We executed iperf3 with several different workload configurations: 1 MB, 10 MB, 100 MB, 1 GB, and 10 GB.

Figure 4(d) shows the energy consumption observed through the execution of the experiments. X-axis represents the bandwidth settings applied in the benchmark. Y-axis represents the energy consumed (in joules) by the server. We observe a smooth increase in the beginning up until 10 Mbps – up to 28% compared to idle. For bandwidths higher than 100 Mbps, we noted a spike in energy consumption, up to 117% compared to idle. Network utilization is, therefore, an important component of energy consumption, depending on its utilization level.

3.6 Container orchestration analysis

So far, this section has analyzed the influence of physical resources in energy consumption. However, large cloud environments comprise thousands of containers that must be properly managed. Our experiments used a Kubernetes [40] deployment comprising a single dedicated master. Even though Kubernetes is a remote service, the objective of this section is to see how customer demands impact the energy consumption behavior of the Kubernetes master.

Figure 5 illustrates this behavior. X-axis represents the number of active containers. The Y-axis represents the energy consumption in joules. Activating the orchestrator corresponds to an increase of 90 J (compared to idle). Adding the first container increases energy consumption to about 390 J. Consumption remains fairly constant up to 32,768 containers.

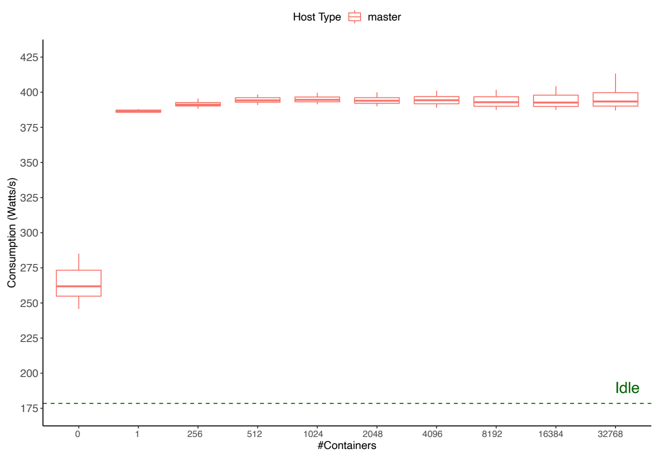


Figure 5 Kubernetes Orchestrator's Energy Consumption.

4 Containers cost model based on energy consumption

Energy costs in the composition of IaaS cloud expenses are undeniable. One of the main contributions of this work is the proposal and description of a pricing model for IaaS providers that considers containers' energy consumption. Tenants are encouraged to optimize their applications when priced based on energy consumption.

The proposed pricing model follows the notation described in Table 2. Initially, P_w represents the price charged by the cloud provider's electrical distribution company, which tax the provider by watts consumed and according to the power's supplement region. W_{idle} describes the energy consumption in watts for a given server in idle status (*i.e.,* without hosting containers). C_{idle} is the server's minimum energy cost, or in other

Table 2 Notation related to the container's energy consumption in a period of time.

Notation	Description
P_w	Price for watts.
W_{idle}	Watts consumption of idle container server.
C_{idle}	Energy costs to maintain active idle server.
T_u	Server's available resources usage rate.
T_c	Server's available resources complement rate.
$\|Cont_u\|$	Tenants containers in a server.
G_p	Providers expenses to maintain available and not allocated resources.
$\|Cont_s\|$	Containers per server.
P_{cpu}	CPU resource price.
W_{cpu}	CPU resources consumed in *watts* by container.
C_{cpu}	Energy consumption related to physical CPU usage.
$U_{cpu}(t, c)$	CPU usage of a specific container c.
W_{mem}	Watts consumed by the container memory resource.
C_{mem}	Energy cost related to memory usage.
(t, c)	Given instant t of a specific container c.
$U_{mem}(t, c)$	Container's memory usage.
fvc	Variation factor of consumption by containers allocated on the server.
C_{net}	Energy costs related to network.
$U_{net}(t, c)$	Network usage of a container.
W_{net}	Watts consumption by network container resources.
C_{disk}	Cost related to input and output storage operations.
P_{disk}	Storage price.
Q_{disk}	Amount of storage used.
W_{orc}	Watts consumed by the container orchestrator.
C_{orc}	Container orchestrator's related energy cost.
$Cost$	Energy cost of a container over a period of time.
C_{total}	Energy cost related to the resources usage by containers of a given customer.

words, is given by minimal energy that the provider expends to keep it active and available for the execution of containers. Each server has a different C_{idle} resulted from hardware-specific characteristics (technologies used, power supplies, CPUs, GPUs, drives, storage, among others). For this reason, the cost of C_{idle} changes depending on the platform on which the containers run. Thus, each specific server has a fixed cost in idle and must distribute it through the proportional apportionment between clients and containers allocated on the server. Therefore, Equation 1 defines the cost to keep the server active.

$$C_{idle} = P_w \times W_{idle} \tag{1}$$

For the proportional share of C_{idle} among the hosted containers, the split of W_{idle} is required. This apportionment consists of 5 elements:

1. T_u denoting the computing resources (the percentage of the resources allocated by clients that run containers on a server);

2. T_c, a fee for complementing the server's computational resources, applied to exempt the customer from incurring a higher cost than effectively allocated and used. This rate equalizes the resource allocation percentage. When a single client allocates a server, and the containers do not use the server as a whole, the consumption of W_{idle} should not be considered 100% for pricing this client. The provider is also responsible for the minimum consumption and assumes a percentage, which we call T_c. Therefore, if the allocated tenants' rate T_u on the platform is less than T_c, the cost's apportionment is carried out between tenants and the provider.

3. G_p, a management cost attributed to the provider due to its responsibility to scale and manage the server computational resources, since keeping a specific machine available is considered a fixed cost;

4. $|Cont_u|$ is the element that counts and assigns the number of the allocated containers by each client on a server;

5. $|Cont_s|$ counts and identifies by unit the allocated containers on the server.

In summary, for a given tenant, the fraction of watts consumption in idle mode (W_{idle}) is given by Equation 2.

$$W_{idle} = \begin{cases} |Cont_u| \times \frac{G_p \times \left(100 + (T_u - T_c)\right)}{|Cont_s|} & \text{if } T_u < T_c \\ |Cont_u| \times \frac{G_p}{|Cont_s|} & \text{else} \end{cases} \tag{2}$$

The Equation 2 verifies if the utilization rate T_u is less than the complementary rate T_c. So the apportionment of this cost occurs between: the number of containers per tenants ($|Cont_u|$); the number of containers per server ($|Cont_s|$); and between the cloud provider (G_p). When T_u is greater than T_c, W_{idle} is apportioned only between $|Cont_u|$ and $|Cont_s|$.

From the identification of the server's cost in idle mode, it is necessary to understand the costs of each resource used by containers (CPU, memory, network, storage, and orchestration).

The CPU price is given by the P_{cpu} element, representing the price paid to the energy distribution company for the server's CPU consumption. This element consists of W_{cpu}, which identifies the number of watts consumed by CPU, and is composed of the elements P_w, and C_{idle}, the watts price and the idle server's cost, respectively, as given by Equation 3.

$$P_{cpu} = (W_{cpu} \times P_w) - C_{idle} \tag{3}$$

Equation 3 enables the formulation of the total CPU cost (C_{cpu}) at a given time interval ($T1, T2$). The total CPU cost is composed of the element $U_{cpu}(t, c)$, which is equivalent to the CPU usage at a given time t of a specific container (c), and by the allocated containers sum per server. So the total CPU cost is expressed in Equation 4.

$$C_{cpu} = \sum_{t=T_1}^{T_2} \sum_{c=1}^{|Cont_u|} U_{cpu}(t, c) \times P_{cpu} \tag{4}$$

Following, W_{mem} reflects the memory consumption's energy, identifying the number of watts consumed by the memory resource. Also, $U_{mem}(t, c)$ is equivalent to the memory usage at a given time t of a specific container (c). Hence, it is possible to identify the energy cost of memory represented by the element C_{men}, addressed in Equation 5, which depicts the memory pricing in the container's execution.

$$C_{mem} = \sum_{t=T_1}^{T_2} \sum_{c=1}^{|Cont_u|} U_{mem}(t, c) \times \Big((W_{mem} \times P_w) - C_{idle}\Big) \tag{5}$$

In turn, the W_{net} accounts for the consumption in watts an application has when using the network resource, highlighting that this resource has little significant variation in the number of active containers on the machine, maintaining a variation only to the network bandwidth used. Therefore, to calculate the element C_{net} it is necessary to analyze two elements in addition to W_{net}: (i) the $U_{net}(t, c)$ and (ii) P_{net}. The element $U_{net}(t, c)$ identifies the network usage at a given time t for a specific container (c), given by the size of the bandwidth used in the container's execution in a given period, through the sum of the time interval, as well as the same-server containers' sum. P_{net} is composed of fvc, which refers to a price variation factor depending on the same-server existing containers' consumption. Therefore P_{net} is detailed in Equation 6, while C_{net} is expressed by Equation 7.

$$P_{net} = \big((W_{rede} \times P_w) - C_i\big) \times fvc \tag{6}$$

$$C_{net} = \sum_{t=T_1}^{T_2} \sum_{c=1}^{|Cont_u|} U_{net}(t,c) \times P_{net} \tag{7}$$

The parameter fvc represents the variation in energy consumption (*i.e.,* even if not very significant, this work understands that it is relevant to assign it to the client), according to the number of containers running on the server. Thus, if the server has only one client running its containers, fvc is 1. If there is more than one client running containers on the server, then the containers' number per client is divided by the total number of containers allocated on the machine and then added 1.

$$fvc = \begin{cases} 1 & \text{if } |Cont_u| = |Cont_s| \\ \frac{|Cont_u|}{|Cont_s|} + 1 & \text{else} \end{cases} \tag{8}$$

Regarding the storage resource, the experimental analyzes (Section 3) demonstrated that the resource is constant in terms of energy consumption when there is only one container running in the server, regardless of the size of the disc I/O. However, when running more than one container per server, the storage's energy consumption is variable. In this sense, the storage cost C_{disk} (described in Equation 9 is composed by U_{disk} (*i.e.,* the storage's size used, and for W_{disk} representing the consumed watts by the storage resource).

$$C_{disk} = \begin{cases} \sum_{t=T_1}^{T_2} \sum_{c=1}^{|Cont_u|} U_{disk}(t,c) \\ \times \big(W_{disk} \times P_w - C_{idle}\big) & \text{if } |Cont_s| > 1 \\ \sum_{t=T_1}^{T_2} U_{disk}(t) \\ \times \big(W_{disk} \times P_w - C_{idle}\big) & \text{else} \end{cases} \tag{9}$$

If the container number is higher than 1, then the storage cost is obtained by double sums. The inner sum considers all containers' storage usage requested by the customer, while the outer sum realizes the sum of the general storage usage in a given interval. Otherwise, the cost is obtained only by the sum of the time interval accounting for the server allocated container's storage usage. In addition to the energy consumption of computational resources in active containers, there is also the container management's energy consumption. Container orchestrating technologies realizes the management of these containers (*i.e.,* Kubernetes, Swarm, Mesos, among others). Therefore, orchestration expenses must be accounted.

The container orchestration analysis (Section 3) reveals the master node is not influenced energetically by the number of managed containers. For this reason, it is necessary that the energy sharing of this management takes place between all the orchestrated containers. The orchestration's energy pricing involves the utilization and compensation fees, i.e., T_u and T_c, respectively. Also, the W_{orc} represents the watts consumed by the orchestrator, defining the cost of container orchestration (C_{orc}). Thus, C_{orc} is defined in Equation 10.

$$C_{orq} = \begin{cases} |Cont_u| \times \frac{\big((W_{orc} \times P_w) - C_{idle}\big) \times \left(\frac{100 + (T_u - T_c)}{100}\right)}{|Cont_s|} \\ \text{if } T_u < T_c \\ |Cont_u| \times \frac{\big[(W_{orq} . P_w) - C_i\big]}{|Cont_s|} & \text{else} \end{cases} \tag{10}$$

Equation 10 determines if the utilization fee T_u is less than the complement fee T_c, then the apportionment of this cost occurs among the number of containers per tenants $|Cont_u|$, and enter the containers number per server $|Cont_s|$. However, if T_u is greater than T_c, the apportionment of this cost occurs only between $|Cont_u|$ and $|Cont_s|$. Thus, if the cloud provider allocates a customer into a machine with no other user, the customer will not pay the full price of the active server. Therefore, Equation 11 presents the container cost, where C_{total} is the cost of using resources per container for a given customer.

$$C_{total} = C_{idle} + C_{cpu} + C_{mem} + C_{rede} + C_{disk} + C_{orc} \tag{11}$$

The proposed cost model is simplified and stratified according to the elements that make up an OS virtualization environment in IaaS cloud providers. Consequently, the calibration of the model components depends on the analysis of the behavior of each resource.

5 AWS Case Study

AWS Fargate [41] is a serverless compute service, allowing container execution without the need to provision servers from customer side. We selected this service due to its pricing model based on on-demand resource allocation. Tenants pay for the number of CPUs allocated, and the amount of memory, storage, and communication (data size) used [42] over time. Tenants can adapt resource demands second by second. Unfortunately, in practice, container management optimization is a complex task. Furthermore, tenants are responsible for monitoring resource utilization and scheduling.

5.1 Energy consumption with Fargate settings

Section 3 depicts the energy consumption analysis of physical resources where the dedicated server has its resources saturated. Since AWS does not allow low-level access to physical resources, we built a similar platform using our Yeti cluster. We used Yeti to reproduce AWS Fargate's set of supported configurations (`https://aws.amazon.com/fargate/pricing/`). We used Docker as the containerization platform and Kubernetes for management since Fargate also uses both technologies. The experimental environment consists of three servers. One server is reserved for the Kubernetes master orchestrator. The others servers run the Docker containers through the Kubernetes workers (Figure 6).

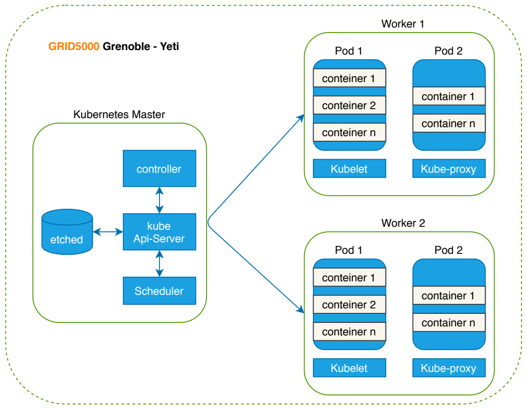


Figure 6 Containers Management. Adapted from [40].

5.1.1 AWS Fargate service offers

Table 3 presents the AWS Fargate offers analyzed. We evaluate energy consumption for each resource at maximum utilization. Our experiments consist of two benchmarks, StressNG and Stream, executing 21 different vCPU and memory consumption scenarios.

Table 3 AWS Fargate service offers. Adapted from [43].

ID	CPU	Memory values
#1	0.25 vCPU	0.5 GB, 1 GB and 2GB
#2	0.5 vCPU	Min. 1GB and Max. 4GB, in 1 GB increments
#3	1 vCPU	Min. 2GB and Max. 8GB, in 1 GB increments
#4	2 vCPU	Min. 4GB and Max. 16GB, in 1GB increments
#5	4 vCPU	Min. 8GB and Max. 30GB, in 1GB increments

VCPU energy consumption is obtained for each ID by keeping memory utilization at idle while stressing CPU. Memory energy consumption is obtained with maximum CPU and memory utilization.

Figure 7(a) depicts energy impact from resources (CPU and memory) based on Fargate service offers. X-axis represents each scenario evaluated. Y-axis represents energy consumption in joules. The dashed green line represents the minimum energy consumption of the server with no workload; *i.e.*, when there is a resource allocation but its applications do not require resources. All scenarios show a gap between the idle line and their energy consumption values. This gap reaches up to 300 joules. However, Fargate does not take this into account when charging customers.

5.1.2 Additional AWS Fargate services

AWS considers storage and data transfers as additional services. For the storage service (Amazon EBS Volumes), tenants have five types of volumes and pay per GB-month of provisioned storage, as described in Table 4. We selected the cold HDD (sc1) volume because it is similar to Yeti's hardware.

Storage devices consume energy to remain in service, regardless of I/O activity. Storage transactions (writes and reads, HDD on our experimets) demand additional energy. Amazon EBS volume service disregards storage transactions and, consequently, the energy consumption

Table 4 Additional AWS offers. Adapted from [44, 45].

Storage Services	Data transfers OUT
General Purpose SSD (gp2) Volumes	Up to 1 GB / Month
Provisioned IOPS SSD (io2) Volumes	Next 9.999 TB / Month
Provisioned IOPS SSD (io1) Volumes	Next 40 TB / Month
Throughput Optimized HDD (st1) Volumes	Next 100 TB / Month
Cold HDD (sc1) Volumes	Greater than 150 TB / Month

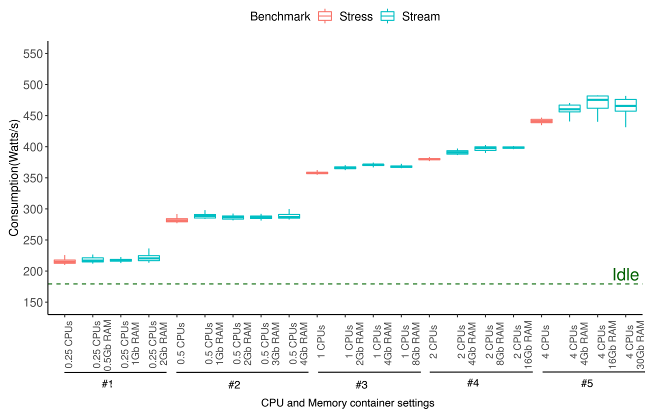
variation. Figure 7(b) depicts storage energy consumption considering disk transaction activities. The X-axis represents the amount of available HDD space to perform HDD-intensive transactions. The Y-axis represents the energy consumed in joules. We used the flexible I/O benchmark [36] to generate HDD-intensive transactions. We varied the number of containers (from one to eight) and HDD space size (60 and 80 GB). The results show that the energy consumption behavior is similar for both space sizes. However, the number of containers has a clear energy impact. For instance, eight containers consume about 70 J more than only one container. Compared to idle (dashed green line, without disk-intensive transactions), the gap reaches 260 J.

Regarding data transfers, Amazon EC2 offers free data transfers into Amazon (inbound). Outbound data transfers are charged according to Table 4. Amazon's data transfer service considers the energy consumption of network equipment to be fixed. However, network consumption may be high in one month and demand no resources in the next month.

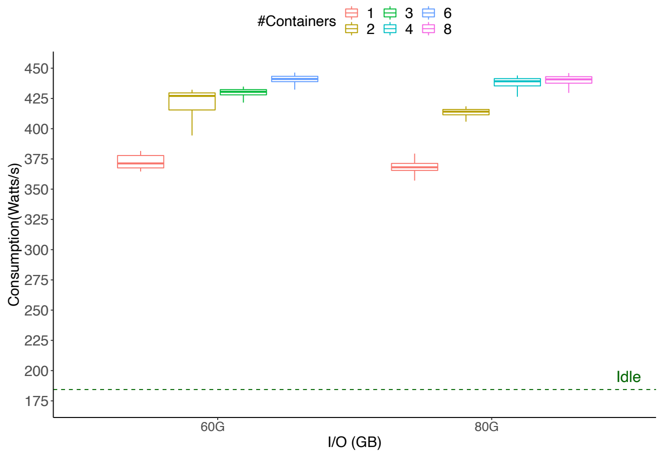
We used iperf3 [37] to generate network load. Figure 8 presents client and server network flow behavior. The X-axis represents the number of containers and bandwidth. The Y-axis represents the energy consumption in joules. We gradually increase network load from 1 MB up to server saturation at 10 GB. Then we oversaturate the server network using loads of 25 GB, 50 GB, and 100 GB to verify its behavior under high traffic. The results show a gradual increase in energy consumption up to 500 MB, with a maximum difference of 50 J when compared to idle server – an increase of 42%. Values higher than 1GB behaved differently, showing non-linear growth in energy consumption and leading the energy consumption gap to a maximum of 270 J compared to the idle – an increase of 157%. If the application aims to reduce energy consumption, it is important to reduce communication between containers It is possible to characterize the energy impact of the relevant storage resource by varying the number of containers that use storage. For the network resource, the energy impact is evident when the resource is saturated.

5.2 EPCC vs. AWS Fargate

This section compares EPCC to AWS Fargate model. The first scenario comprises two configurations with two and four CPUs.



(a) CPU and Memory mixed energy consumption.



(b) Storage energy consumption.

Figure 7 Experiment results.

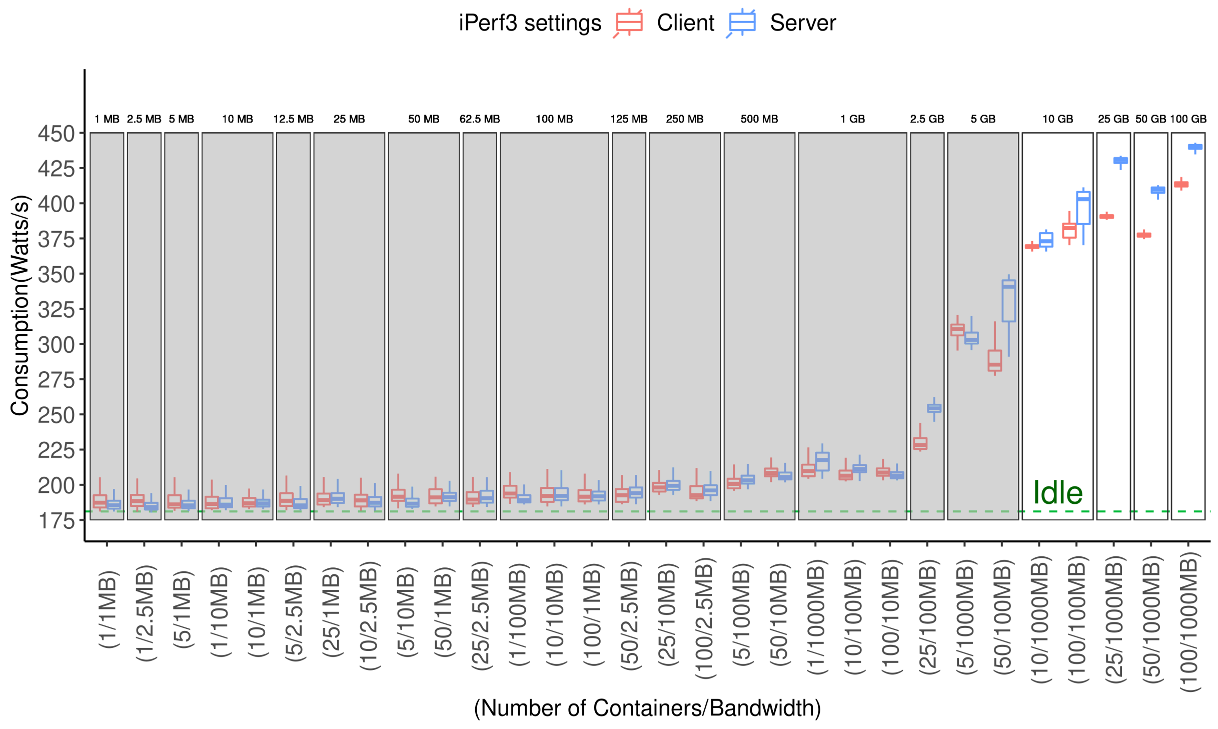


Figure 8 Network Energy Consumption.

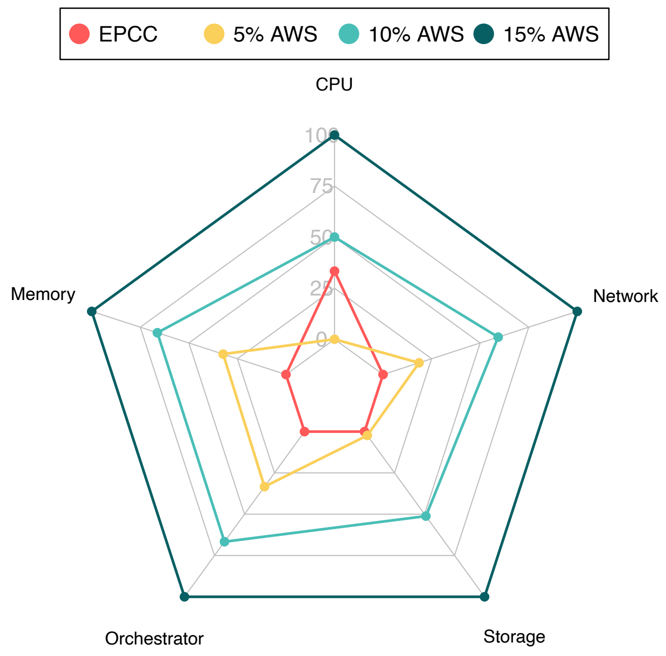
Table 5 AWS Fargate Price.

CPUs	Usage (%)	Price (US$/hour)
2	50%	0.08
	100%	0.08
4	50%	0.16
	100%	0.16

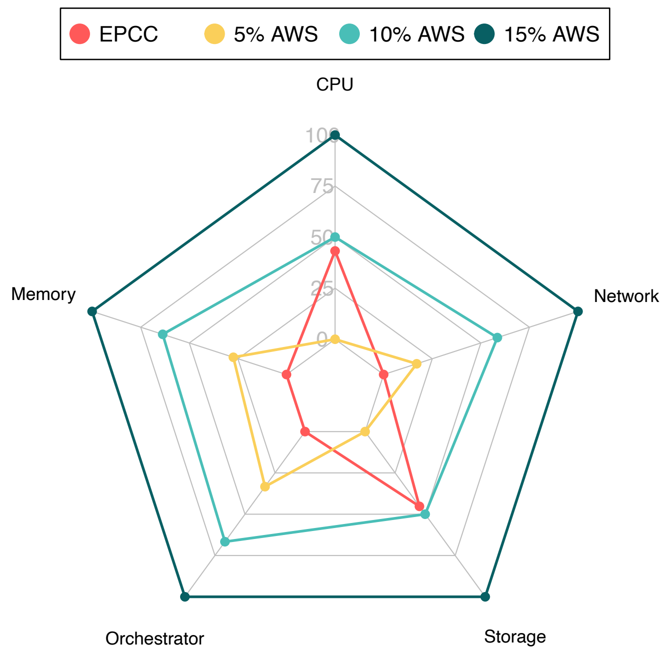
Table 5 reveals that Fargate only considers the amount of CPU and memory provisioned, not their actual usage (percentages). The prices, therefore, are the same for 50% and 100% CPU resource utilization. Nevertheless, one CPU (corresponding 50% of CPU provisioned demand) represents 5.57% less energy consumption than the 100% considered by AWS Fargate. The customer's bill must consider this difference. Since containers leverage a flexible platform, the cost model should also be flexible. A flexible cost model motivates customers to make their applications more efficient, encouraging energy savings and green IT practices.

5.2.1 *Estimated price of individual cost components*

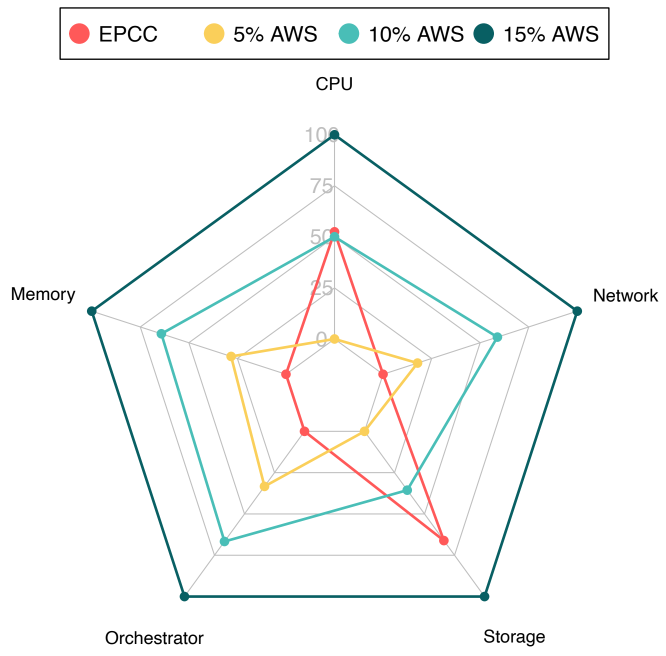
EPCC considers five components' energy consumption to calculate the cost: CPU, memory, storage, network,



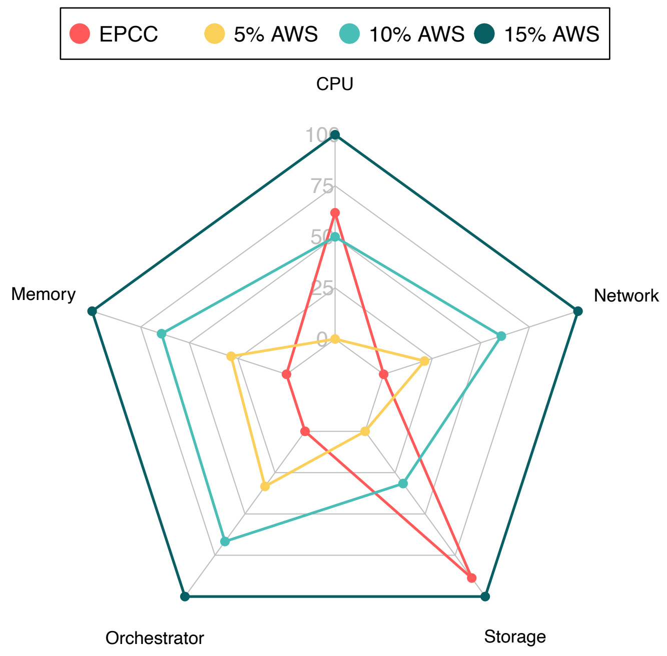
(a) Component use by 25%.



(b) Component use by 50%.



(c) Component use by 75%.



(d) Component use by 100%.

Figure 9 EPCC *vs.* AWS Fargate: Estimated price of individual cost components according to usage.

and container orchestration. In contrast, since AWS has a private cost model, the proportion of the energy cost in the final customer's billing is unknown to us, and we cannot identify these publicly available data. The literature suggests that energy costs reach up 50% of total DC costs [9]. Operating costs include air conditioning and other energy-intensive equipment. In PSVE [11], the authors estimate that energy consumption can be between 5 and 15% of IaaS instance price. Our work follows this estimate. Also, we adopt the energy price of US$ 0.0773 per kWh (This is the value from the electrical distributor in the northern Virginia region in the United States, where the AWS DC is located).

Figure 9 presents estimated prices. As EPCC considers components' usage, the results show the estimated price for four different scenarios (25%, 50%, 75%, and 100%). Each radio graph has a proportional axis, in which the maximum value is 100%, and the minimum is 0%. Therefore, the highest prices are on edge and the lowest one in the center. Each axis represents the estimated price, with energy cost to 5%, 10%, 15%, and EPCC, for each cost model component (CPU, memory, network, storage, and orchestrator). We represented the scenarios using different colors using solid lines linked by dashes, forming a polygonal surface. The analysis of combined components is realized by calculating the polygon's surface area. The smaller the surface area, the lower the price.

In all scenarios (Figure 9 (a)-(d)), the AWS 15% estimate (dark green) is the worst case, as expected, concerning price (highest). Although AWS 10% (light green) is better than EPCC when CPU and storage usages are 75% and 100%, the surface area of EPCC remains smaller than the AWS 10% estimate' surface area. Finally, comparing the AWS 5% (orange estimate) and EPCC, EPCC has at least three of five dots in the center (the best result), but it has an estimate price higher than AWS 5% on two scenarios (75% and 100%). As expected, EPCC is best when component use is low. More importantly, the AWS pricing model cannot reflect lower utilization of energy, which corroborates to our main point the importance of aggregating financial benefit to engender energy savings.

The total price considers energy cost to keep an active idle server in addition to the individual component prices. Figure 10 shows a radio graph including five different scenarios based on components usage; *e.g.,* 0% shows idle server, and for 100% all components are saturated. For all scenarios, EPCC leads to total prices lower than AWS 15% and 10%. The AWS 5% estimated has the lowest value for the total price, except when the server is idle. Finally, the price estimates from EPCC adapt to energy consumption according to components usage. Thus, the tenants are encouraged to improve energy consumption, optimize resource utilization, and apply green IT concepts.

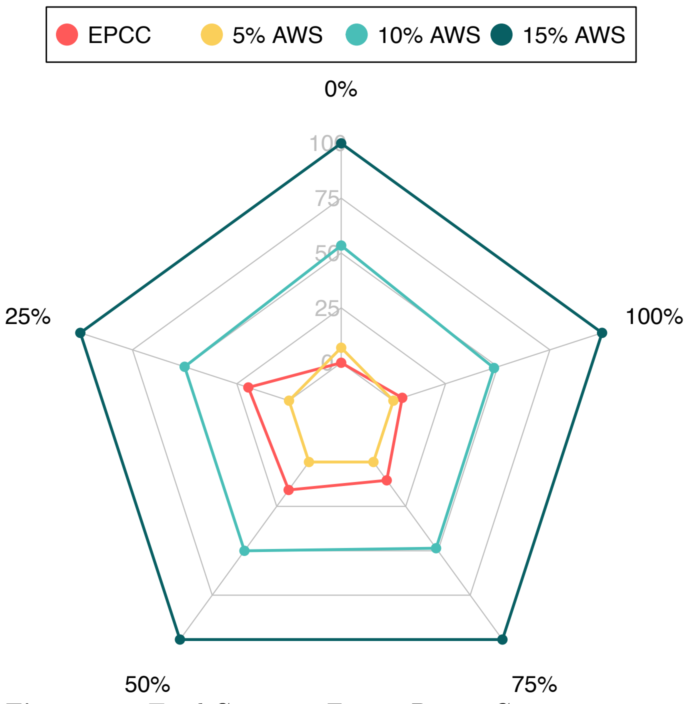


Figure 10 Total Container Energy Price - C_{total}.

6 Considerations and Future works

Energy is one of the main components of data centers' total cost and physical infrastructure in cloud computing. Since the emergence of cloud computing, cost models have evolved while service offers expanded. Among other characteristics, resource optimization is one of the key characteristics of the cloud service model. Cloud tenants are aware of where resources are allocated because it directly impacts their bills. However, energy still is missing in this equation, especially since its impact is not perceived by tenants who are not encouraged to optimize its utilization.

This work proposed EPCC, a cost model that rewards energy awareness by accounting for the usage of allocated resources. EPCC combines the energy component with the allocation component to determine where and how to contract services and use resources. Our analysis of individual resources and their energy consumption reveals that an application with resource-intensive demands can consume 3-4x more energy. Finally, a resource-intensive application can count up to US$ 0.049/hour more than an idle one.

Considering the limitations identified during the realization of this research, it was observed the possibility of considering in a future work a detailed analysis of the Kubernetes energy consumption. Since in the EPCC proposal, the Kubernetes master was deployed in an isolated structure (i.e., the only service running on the VM). Thus, this specific energy consumption of each computational resource (i.e., network, memory, storage, and CPU) was not measured.

Another point to be measured in future work is the apportionment of the energy consumption of different virtualization services hosted on the same host node. For example, when running a VM and a container in parallel on the same host node, then how should the energy cost of computational resources be attributed to the different environments?

Future work will also comprises expanding the research to evaluate the impact of resource utilization on energy consumption using other cloud providers and services, such as OpenStack and its services: Nova, Magnum, and Ironic.

Acknowledgement

Blind review version.

References

[1] Prateek Sharma, Lucas Chaufournier, Prashant Shenoy, and YC Tay. Containers and virtual machines at scale: A comparative study. In *Proceedings of the 17th International Middleware Conference*, page 1. ACM, 2016.

[2] Murugiah Souppaya, John Morello, and Karen Scarfone. Application container security guide. *NIST Special Publication*, 800:190, 2017.

[3] GB Hima Bindu, K Ramani, and C Shoba Bindu. Energy aware multi objective genetic algorithm for task scheduling in cloud computing. *International Journal of Internet Protocol Technology*, 11(4):242–249, 2018.

[4] Saira Begum and Muhammad Khalid Khan. Potential of cloud computing architecture. In *Information and Communication Technologies (ICICT), 2011 Int. Conf. on*, pages 1–5. IEEE, 2011.

[5] Louis Columbus. State of cloud adoption and security. *Forbes*, 2017.

[6] Luiz F Bittencourt, Alfredo Goldman, Edmundo RM Madeira, Nelson LS da Fonseca, and Rizos Sakellariou. Scheduling in distributed systems: A cloud computing perspective. *Computer Science Review*, 30:31–54, 2018.

[7] Caesar Wu, Rajkumar Buyya, and Kotagiri Ramamohanarao. Cloud pricing models: Taxonomy, survey, and interdisciplinary challenges. *ACM Computing Surveys (CSUR)*, 52(6):108, 2019.

[8] Radoslav Danilak. Why energy is a big and rapidly growing problem for data centers. 2017, 2017.

[9] T Comerford. How data center operators can avoid energy price hikes this winter. *Data Center Knowledge*, 2015.

[10] Lucas Litter Mentz, Wilton Loch, and Guilherme Koslovski. Comparative experimental analysis of docker container networking drivers. In *9th IEEE International Conference on Cloud Networking (IEEE CloudNet 2020)*, november 2020.

[11] Mauro Hinz, Guilherme Piegas Koslovski, Charles C Miers, Laércio L Pilla, and Maurício A Pillon. A cost model for iaas clouds based on virtual machine energy consumption. *Journal of Grid Computing*, 16(3):493–512, 2018.

[12] Mascha Kurpicz, Anne-Cécile Orgerie, Anita Sobe, and Pascal Felber. Energy-proportional profiling and accounting in heterogeneous virtualized environments. *Sustainable Computing: Informatics and Systems*, 18:175–185, 2018.

[13] Tom Bawden. Global warming: Data centres to consume three times as much energy in next decade, experts warn. *The Independent*, 23, 2016.

[14] Juanli Hu, Jiabin Deng, and Juebo Wu. A green private cloud architecture with global collaboration. *Telecommunication Systems*, 52(2):1269–1279, 2013.

[15] S. Pandikumar, S. P. Kabilan, and L. Amalraj. Article: Green it: A study and analysis of environmental impact of social networks and search engines. *International Journal of Computer Applications*, 60(6):–, December 2012. Full text available.

[16] A. Jain, M. Mishra, S. K. Peddoju, and N. Jain. Energy efficient computing- green cloud computing. In *2013 International Conference on Energy Efficient Technologies for Sustainability*, pages 978–982, April 2013.

[17] Jianhui Zhang, Keqiu Li, Deke Guo, Heng Qi, Haisheng Yu, Yingwei Jin, and Arun Kumar Sangaiah. Sustainable green data center: Guaranteeing flow deadlines in chains of virtual network functions with mrouting. *Sustainable Computing: Informatics and Systems*, 2018.

[18] S.K. Garg and R Buyya. Green cloud computing and environmental sustainability. pages 315–340, 01 2012.

[19] P. Sharma, P. Pegus II, D. Irwin, P. Shenoy, J. Goodhue, and J. Culbert. Design and operational analysis of a green data center. *IEEE Internet Computing*, pages 1–1, 2017.

[20] Jordi Guitart. Toward sustainable data centers: a comprehensive energy management strategy. *Computing*, 99(6):597–615, 2017.

[21] C. G. Kominos, N. Seyvet, and K. Vandikas. Bare-metal, virtual machines and containers in openstack. In *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, pages 36–43, March 2017.

[22] Peter Mell, Tim Grance, et al. The nist definition of cloud computing. 2011.

[23] Ali Hammadi and Lotfi Mhamdi. A survey on architectures and energy efficiency in data center networks. *Computer Communications*, 40:1–21, 2014.

[24] Vitor Goncalves da Silva, Marite Kirikova, and Gundars Alksnis. Containers for virtualization: An overview. *Applied Computer Systems*, 23(1):21–27, 2018.

[25] Zachary J Estrada, Zachary Stephens, Cuong Pham, Zbigniew Kalbarczyk, and Ravishankar K Iyer. A performance evaluation of sequence alignment software in virtualized environments. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, pages 730–737. IEEE, 2014.

[26] Nam Yong Kim, Jung Hyun Ryu, Byoung Wook Kwon, Yi Pan, and Jong Hyuk Park. Cf-cloudorch: container fog node-based cloud orchestration for iot networks. *The Journal of Supercomputing*, 74(12):7024–7045, 2018.

[27] Susan Moore. Gartner forecasts strong revenue growth for global container management software and services through 2024. `https://www.gartner.com/en/newsroom/press-releases/2020-06-25-gartner-forecasts-strong-revenue-growth-for-global-co`, 2020.

[28] Arka A Bhattacharya, David Culler, Aman Kansal, Sriram Govindan, and Sriram Sankar. The need for speed and stability in data center power capping. *Sustainable Computing: Informatics and Systems*, 3(3):183–193, 2013.

[29] Muhammad Zakarya and Lee Gillam. Managing energy, performance and cost in large scale heterogeneous datacenters using migrations. *Future Generation Computer Systems*, 2018.

[30] Rolando Brondolin, Tommaso Sardelli, and Marco D Santambrogio. Deep-mon: Dynamic and energy efficient power monitoring for container-based infrastructures. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 676–684. IEEE, 2018.

[31] Philipp Leitner, Jürgen Cito, and Emanuel Stöckli. Modelling and managing deployment costs of microservice-based cloud applications. In *Proceedings of the 9th International Conference on Utility and Cloud Computing*, pages 165–174. ACM, 2016.

[32] DIAMANTI. Container adoption benchmark survey. Technical report, 2019.

[33] GRID5000. GRID5000 large-scale and flexible testbed for experiment-driven research, October 2020. `https://www.grid5000.fr/`.

[34] Colin Ian King. Stress-ng, Oct 2019.

[35] John D McCalpin et al. Memory bandwidth and machine balance in current high performance computers. *IEEE computer society technical committee on computer architecture (TCCA) newsletter*, 2(19–25), 1995.

[36] Jens Axboe. 1. fio - flexible i/o tester rev. 3.23¶, 2017.

[37] Mathijs Mortimer. iperf3 documentation, 2018.

[38] Zheng Li, Selome Tesfatsion, Saeed Bastani, Ahmed Ali-Eldin, Erik Elmroth, Maria Kihl, and Rajiv Ranjan. A survey on modeling energy consumption of cloud applications: deconstruction, state of the art, and trade-off debates. *IEEE Transactions on Sustainable Computing*, 2(3):255–274, 2017.

[39] Roberto Morabito. Power consumption of virtualization technologies: an empirical investigation. In *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*, pages 522–527. IEEE, 2015.

[40] Kubernetes. Kubernetes components. `https://kubernetes.io/docs/concepts/overview/components/`, 2020.

[41] Fargate AWS. AWS Fargate serverless compute for containers, October 2020. `https://aws.amazon.com/en/fargate/`.

[42] Deepak Vohra. *Amazon Fargate Quick Start Guide: Learn how to use AWS Fargate to run containers with ease*. Packt Publishing Ltd, 2018.

[43] Amazon Web Services (AWS). AWS Fargate Pricing, October 2020. `https://aws.amazon.com/fargate/pricing/`.

[44] Amazon Web Services (AWS). Amazon EBS Volumes, October 2020. `https://aws.amazon.com/fargate/pricing/`.

[45] Amazon Web Services (AWS). Data Transfer, October 2020. `https://aws.amazon.com/ec2/pricing/on-demand/#Data_Transfer`.