computers

MDPI

Article

Improving Efficiency of Edge Computing Infrastructures through Orchestration Models †

Raffaele Bolla [1], Alessandro Carrega [2], Matteo Repetto [2,*] and Giorgio Robino [2]

[1] Electrical, Electronics and Telecommunication Engineering and Naval Architecture Department (DITEN), University of Genoa, Via Opera Pia 13, 16145 Genova, Italy; raffaele.bolla@unige.it

[2] S3ITI Lab, National Inter-University Consortium for Telecommunications (CNIT), Via Opera Pia 13, 16145 Genova, Italy; alessandro.carrega@cnit.it (A.C.); giorgio.robino@cnit.it (G.R.)

* Correspondence: matteo.repetto@cnit.it; Tel.: +39-010-353-2057

† This paper is an extended version of our paper published in Carrega, A.; Portomauro, G.; Repetto, M.; Robino, G. OpenStack extensions for QoS and energy efficiency in edge computing. In Proceedings of the 3rd IEEE International Conference on Fog and Edge Mobile Computing (FMEC 2018), Barcelona, Spain, 23–26 April 2018.

Received: 22 May 2018; Accepted: 14 June 2018; Published: 20 June 2018

Abstract: Edge computing is an effective paradigm for proximity in computation, but must inexorably face mobility issues and traffic fluctuations. While software orchestration may provide effective service handover between different edge infrastructures, seamless operation with negligible service disruption necessarily requires pre-provisioning and the need to leave some network functions idle for most of the time, which eventually results in large energy waste and poor efficiency. Existing consolidation algorithms are largely ineffective in these conditions because they lack *context*, i.e., the knowledge of which resources are effectively used and which ones are just provisioned for other purposes (i.e., redundancy, resilience, scaling, migration). Though the concept is rather straightforward, its feasibility in real environments must be demonstrated. Motivated by the lack of energy-efficiency mechanisms in cloud management software, we have developed a set of extensions to OpenStack for power management and Quality of Service, explicitly targeting the introduction of more context for applications. In this paper, we briefly describe the overall architecture and evaluate its efficiency and effectiveness. We analyze performance metrics and their relationship with power consumption, hence extending the analysis to specific aspects that cannot be investigated by software simulations. We also show how the usage of context information can greatly improve the effectiveness of workload consolidation in terms of energy saving.

Keywords: energy efficiency; QoS; edge computing

1. Introduction

Edge computing is an effective solution to tackle a number of challenging requirements that cannot be satisfied by the legacy cloud paradigm (e.g., geographical distribution, computing proximity, transmission latency). It represents an essential building block in the 5G vision, in order to create large distributed, pervasive, heterogeneous, and multi-domain environments [1]. Edge computing will be an integral part of smart orchestration platforms specifically designed to fulfill the challenging requirements of many vertical industries: smart manufacturing, energy, e-health, automotive, media and entertainment [2].

Differently from the cloud, where resources are located in one or a few big data centers and the workload may be quite predictable, edge computing will inexorably face mobility issues, similar to those already present in cellular networks. With the growth of user-centric services, characterized by interactivity and strict constraints on latency, the workload is expected to follow user distribution and mobility patterns, hence leading to uneven, unsteady, and non-uniform distribution within the network. Consequently, the usage of peripheral installations is expected to dynamically vary with hourly, daily, weekly, and seasonal periodicity.

The heterogeneity in hardware virtualization (especially the hypervisor) does not allow for migrating virtual functions between different installations. To cope with this limitation, proper management hooks can be developed for each specific virtual function to enable *state* migration between two running instances in different infrastructures. Software-defined networking easily allows duplication of traffic flows during the migration process, in order to avoid packet losses and timeout expiration. However, the time to provision resources is not negligible because of technical and administrative issues; indicatively, it may take from minutes to days to have a Virtual Machine (VM) ready. Hence, the alternative for mobility management is either reactive provisioning after need, with a delay that might not be acceptable for most demanding applications, or proactive provisioning, which implies running idle instances that might never even be used.

Proactive provisioning basically consists in overprovisioning. It has been the common 'least-effort' relief for long, but it leads to poor utilization of the overall system, raises power consumption and crumbles the overall efficiency, hence it is no more acceptable today in its current form, both for environmental and economical reasons. The real problem is the need to run idle devices, which results in poor efficiency in terms of power consumption per actual computation load [3]. Such inefficiency is also intrinsic in the cloud paradigm, but it is far more severe in edge computing due to more variable workload and larger number of installations.

Despite the plethora of energy-efficiency algorithms already proposed in the literature, we argue that all of them operate in a "blind" way, i.e., they are not aware of the different role applications may have in a business process. As a matter of fact, the majority of applications will probably be active and working most of the the time, but some others might be present just as backup replicas for redundancy or scaling purposes. In some cases (e.g., billing from utilities), some processes run intensively only a few days per month, and remain totally idle the rest of the time. When each application is deployed in a separate VM (as often happen today), consolidation algorithms might be greatly enhanced in both efficiency and Quality of Service if an explicit indication about the usage pattern were available.

The main novelty of our work consists in exploring this opportunity. Roughly speaking, we think that unused VMs may be put in special states by the user (equivalent to suspending laptops by closing their lid), and this information could be used to selectively apply more aggressive power-saving mechanisms to servers that only host unused VMs. In more abstract terms, this means more 'context' is inserted in the consolidation process. Though frequent and timely changes in VMs state may be cumbersome for humans, the birth of software orchestration tools (e.g., OpenBaton [4]) allows for easily automating these operations with other relevant life-cycle management actions (i.e., scaling, backup, resilience).

In a recent paper [5], we have already addressed the design of efficient virtualization frameworks for edge computing, balancing Quality of Service (QoS) with Energy Efficiency (EE). We have proposed a set of functional and semantic extensions to OpenStack, a well-known cloud management software, which enable better management of QoS and EE. Our previous work provided very limited experimental validation, especially concerning energy efficiency. Indeed, packet loss and jitter were measured during network re-configuration (changes in forwarding paths), and power profiles of servers and switches were collected. In this paper, we build on this experimental infrastructure to analyze its effectiveness (i.e., energy saving) and efficiency (i.e., useful work per watt), which are aspects that cannot be realistically investigated by simulations. We include deeper investigation on the relationship between placement strategies and power consumption; we also show how, even with very simple consolidation strategies, the knowledge of context information allows better results than existing practice. Finally, we demonstrate that our approach achieves better linear relationship between usage and power consumption.

The paper is organized as follows. We review related work in Section 2. We better explain the main motivations behind our work, and the relationship between service orchestration and workload consolidation in Section 3. We then describe the implementation of the energy-efficient infrastructure, which provides support for QoS and power management in Section 4. We report evaluation and numerical analysis from the testbed in Section 5. Finally, we give our conclusions in Section 6.

2. Related Work

Energy efficiency of cloud infrastructures implies that the least amount of energy be used for actual computation. This is largely dependent on a more linear relationship between power and performance [6], which is usually pursued by leveraging existing power saving mechanisms: performance scaling (e.g., dynamic regulation of voltage/frequency for CPUs), low-power idle, and, most of all, removal of idle periods. Indeed, running idle servers wastes large amounts of energy to keep the system up and running, without carrying out any useful work. Under this premise, running VMs on the smallest number of servers, while either powering down or putting to sleep all unused devices, is often the most effective solution to save energy. The process of clustering VMs together and selecting the active servers is conventionally indicated as workload 'consolidation'.

Similar to our purpose, Shirayanagi et al. [7] considered the presence of backup replicas and links, and investigated simulations about how power consumption changes when varying their number. However, the focus was on the power consumption of the network alone. Li et al. [8] investigated the impact of overprovisioning (e.g., for dealing with sudden peaks of workload) on power consumption. Their strategy for workload consolidation only considers the servers, and did not take into account any network metric. In this case, they provided a real implementation, based on their virtualization infrastructure named iVIC. Evaluation on violations of service-level agreements in case of consolidation was carried out by Buyya et al. [9], again through simulations.

Despite the large number of algorithms proposed in the literature, the feasibility, effectiveness, and efficiency of workload consolidation in real testbeds have not been fully investigated yet. Heller et al. [10] analyzed how redundancy and guard margins influence the power consumption; their work places VMs on servers with the objective to minimize the power consumption of the data center network, but does not consider the energy drawn by computing servers. Since the grounds for any consolidation algorithm is the capability to move VMs between servers, Voorsluys et al. [11] studied the impact of live migration on application response time.

More recently, Beloglazov et al. [12] proposed OpenStack Neat for balancing QoS and energy consumption. Their framework monitors CPU utilization by VMs, estimates overloading/underloading conditions on each server, and uses OpenStack Application Programming Interfaces (APIs) for live-migration of VMs between servers. Their consolidation strategy is power-unaware, since only the utilization metric is used. Even if power saving should stem from putting idle servers to sleep, their hardware did not support this state, so they only computed estimations for energy saving. Compared to our approach, Neat does not include a power monitoring framework, software-defined networking, and specific context for applications. Rossigneux et al. [13] implemented Kwapi, which is an OpenStack extension to collect real-time power measurements from servers. We used this framework in our implementation [14], and enriched it by also collecting power consumption from networking equipment. Kwapi was also used by Cima et al. [15] in their implementation of OpenStack extensions for energy efficiency. This work has very similar objectives to ours: an energy-efficient cloud infrastructure with enhanced API for power management. The API includes specific commands to change the power state of servers (sleep/active). The authors measured migration times and service downtime for live migration, and energy saving with power-unaware consolidation strategies similar to Neat. Our work follows a similar approach, but we also include software-defined networking for bandwidth reservation, and context information for VMs, which enables better QoS management and more aggressive consolidation strategies.

3. A Paradigm for Energy-Efficient Computing

Fed by the outbreak of cloud services, the usage of data centers is still expected to increase in the next years. Data centers are already responsible for a large share in overall energy consumption (around 2%) [16], so energy saving is necessary among the top management priorities.

The introduction of more efficient hardware, power-saving mechanisms, and more energy-efficient infrastructure design has notably reduced previous forecasts on energy consumption, but the still increasing trend motivates additional effort. As a matter of fact, recent reports on the sustainability of data centers show that larger installations are already approaching ideal effectiveness, while there are still large margins of improvement for small installations [16,17], like those expected for edge computing.

In particular, the real challenge to tackle is power consumption per useful computing work. In this respect, we believe that sharing more context information between the service and the virtualization infrastructure represents a necessary evolution to boost far more efficiency than today, effectively balancing performance, QoS, and power consumption.

3.1. Efficiency vs. Effectiveness

The prevailing optimization target for energy-efficiency in data centers is usually *Power Usage Effectiveness* (PUE), which is defined as the ratio between total facility energy (also including cooling, lightning, power distribution losses) and IT equipment energy. The actual significance of this parameter is still controversial, though, because it does not account for efficiency of IT equipment [18,19]. For example, in case of low server utilization, the amount of power drawn to keep up and running the whole infrastructure is disproportionate to the actual computation service. Workload consolidation explicitly pursues *Computing Usage Efficiency*, by chasing a more linear relationship between power consumption and infrastructure usage.

The common weakness of most consolidation algorithms is to take into account static metrics only (i.e., CPU, RAM, bandwidth demand), and to neglect the current utilization of resources. Low utilization is most likely to happen with variable workload; this is a typical situation for billing applications, video streaming, and many other relevant services which are used with more or less predictable periodic patterns. As we already discussed, such applications are typically sized to deal with peak load, in order to avoid service disruption and delay caused by technical and administrative provisioning procedures.

Tackling more linearity between power consumption and effective data center productivity, some recent proposals consider CPU utilization as main metric for consolidation [12,15]. This approach achieves better efficiency, but it needs some time to detect overloading conditions, hence leading to potential violations of the Service Level Agreement (SLA). Indeed, we think that service users should be aware of any potential performance violations, and should be rewarded for their willingness to participate in more aggressive power-saving mechanisms (for instance, by dynamic and context-aware pricing schemes [20]).

3.2. More Context for Edge Applications

Our energy-efficiency scheme stems from the availability of novel software development and orchestration paradigms for cloud-based applications, as TOSCA [21] and ETSI MANO [22]. Such paradigms build complex applications as logical topologies (*service graphs*) of elementary components (*nodes* or *virtual functions*) that are deployed in independent virtual machines or containers. These models make extensive usage of metadata and annotations to drive the (semi-)automated orchestration process; our idea is to include specific context information concerning QoS and energy efficiency, which can be used to design advanced consolidation strategies [14].

Specifically, we consider the following kinds of context information for each VM:

- *service level*, related to the criticality of the service and the expected QoS (CPU load, network bandwidth);
- *availability*, indicating whether the component will be used or not in the next timeframe.

We model the above information by a colored label associated with each VM:

- *red*, for those VMs that require strict commitment on processing, memory, availability, and bandwidth requirements;
- *yellow*, for those VMs that do not operate close to their performance limit, so there is some degree of flexibility on QoS constraints;
- *green*, for unused (idle) VMs

Red labels are assigned to VMs that host the most critical software, with strict QoS constraints. As compensation for higher fares, users expect no overcommitment and no live migration to occur in this case. For instance, full CPU usage is always assumed, while network bandwidth is reserved according to the maximum declared throughput. Yellow labels are assigned to standard VMs, for which overcommitment is acceptable. In this case, CPU usage may be assumed equal to a declared average load, while network reservation may use mean or equivalent bandwidth values. Finally, green labels are assigned to idle and unused VMs. They can be subject to extreme overcommitment and aggressive power-saving mechanisms, provided they return available with very low delay (i.e., below a few seconds).

In addition to labeling, we also include max/average CPU utilization and network throughput requirements as metadata available in the service graph. This classification could be easily extended to more classes and a richer semantics for QoS.

4. Energy-Efficient Infrastructure

Our energy-efficient infrastructure builds on and extends previous work with similar objectives [12,13,15]. The target is to design a framework that is agnostic of any specific orchestration model and consolidation algorithm. In this respect, it is conceived to extend existing management software by allowing better control on network traffic and power management. Figure 1 shows the architecture of our energy-efficient infrastructure, which includes:

- computing hardware and hypervisors, with power-saving mechanisms;
- cloud management software (CMS) that implements the Infrastructure-as-a-Service (IaaS) model;
- software-defined network (SDN), fully programmable for optimizing network paths according to bandwidth requests; power-saving mechanisms are also available in network devices;
- monitoring, which collects measurements about power consumption and CPU utilization from all servers and network devices;
- management interfaces, which allow interaction both for using and managing the infrastructure.

Our Energy-Efficient infrastructure implements the Infrastructure-as-a-Service model and provides two kinds of interfaces: North and East.

The North interface is conceived to deploy and run applications. It includes all APIs to create, destroy, and manage cloud resources; in addition, we envision additional semantics to specify the context information discussed in Section 3.2. Though these APIs could be used by humans, we expect an orchestration tool takes care of automating the application life-cycle.

The East interface provides access to control and management features. In addition to typical administrative operations in cloud management software (e.g., user management, provider networks, hypervisor management, VM migrations), we also include power-management and SDN capabilities. As depicted in Figure 1, the typical usage of this interface is by consolidation algorithms. An example of a novel algorithm that makes use of context information is described in our previous work [14].

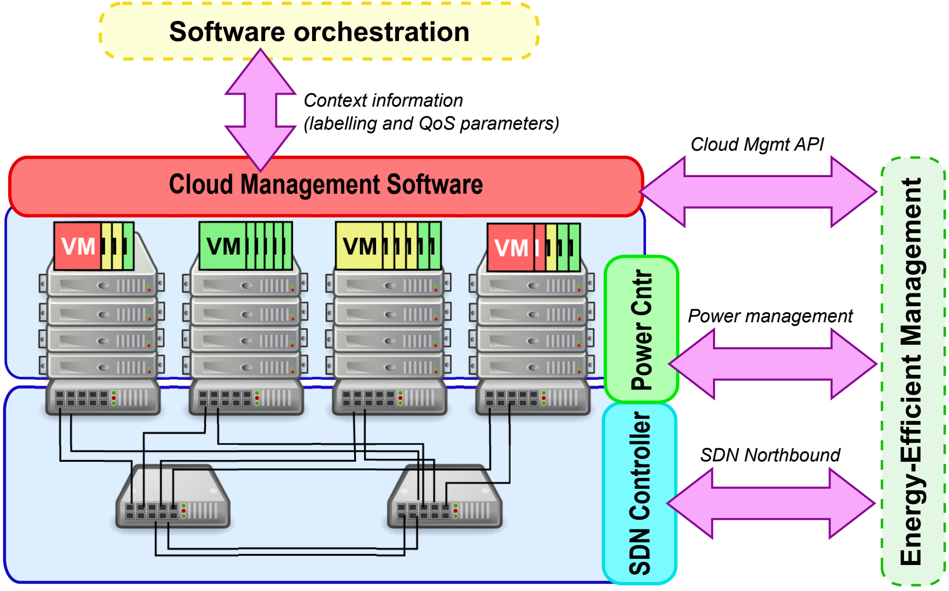


Figure 1. Architecture of the energy-efficient infrastructure.

4.1. Computing Servers

Computing servers support ACPI power states and optionally other power-saving mechanisms (dynamic voltage/frequency scaling, low-power idle). Currently, we only use ACPI S3 state (aka 'Standby', 'Sleep', or 'Suspend-to-RAM'). This is one of the most useful power states, since it almost cuts off all power, while allowing for restoring full operation in a few hundred milliseconds. We plan to include additional controls over other power-saving saving mechanisms in future updates. For what concerns remote control of these states, a Power State Management Profile is already available in the DTMF Common Information Model [23] and mapped to IPMI [24], but it is not available in most commercial devices. While lacking broad support for specific interfaces, a custom API is currently used to change the power state from ACPI S0 to S3 (i.e., to put the device to sleep), whereas Wake-on-Lan is used to resume it back to full operation (from ACPI S3 to S0).

Our installation uses the QEMU/KVM hypervisor, which is the default choice for OpenStack (see below).

4.2. Cloud Management Software

We use the open-source OpenStack framework as cloud management software. Our installation includes authentication (Keystone), computing (Nova), networking (Neutron), image (Glance), storage (Cinder), and telemetry (Ceilometer) modules. The Neutron ML2 plugin is used for virtual networking, together with openvswitch software-based switch. VLAN encapsulation is used for tenant networks because this is the only tunneling protocol managed by OpenFlow in physical switches. The network node is installed on a dedicated machine. A shared file system is available on all compute nodes, so that live-migration of VMs is possible between hypervisors.

4.3. Software-Defined Network

The Software-Defined Network is composed of OpenFlow software switches, running openvswitch. We preferred software switches running on commercial desktops over commercial hardware because the latter does not provide ACPI power states, so cannot be used to conduct live tests on power consumption. In addition to OpenFlow, network devices also expose the GAL interface (Green Abstraction Layer [25]), a recently introduced standard to report and modify energy-related characteristics (power states, power/performance relationship, etc.). The GAL is used to collect the power profile of each device (e.g., the description of how energy consumption changes with different utilization levels), and to change the power state from active (ACPI S0) to sleep (ACPI S3). Devices are resumed back to full operation (from ACPI S3 to S0) by the Wake-on-Lan protocol.

OpenDayLight is used as network controller. We use the l2switch feature to provide full connectivity among all nodes, while explicit flows are configured between VMs with bandwidth requirements. This approach limits the number of OpenFlow rules to specific flows with QoS requirements (which will be mapped to higher priority traffic classes), while falling back to best effort flooding behavior for ancillary traffic (e.g., DNS queries and DHCP).

The l2switch uses controlled flooding to forward packets, so it generates a large amount of network traffic, but it is extremely resilient to failures and does not require reconfiguration in the case of VM migration. Instead, explicit configuration of paths results in better and more efficient usage of the infrastructure, but needs reconfiguration in the case of migration/failure that may lead to performance degradation and service disruption. There aspects are explicitly considered in our performance evaluation (see Section 5.2).

Configuration of QoS parameters (priority queues, traffic shapers, scheduling disciplines) in the physical switches is not implemented yet, but it is already on the development roadmap.

4.4. Monitoring Framework

The main objective for the monitoring component is the collection of measurements from heterogeneous devices, by retrieving data from different meters, with different communication protocols, and with different data formats. To this aim, we use Kwapi [13], a modular framework for monitoring power consumption and publishing data in an OpenStack Ceilometer. We extended Kwapi to collect data about CPU usage of the hypervisors, so we can build the power profile of each device, i.e., the relationship between power consumption and performance [26].

Kwapi includes drivers to query commodity power meters over SNMP, serial/usb wattmeters, and the IPMI interface available in many recent computing boards; this covers most hardware that could be deployed in edge sites, including legacy servers with a single motherboard as well as blade servers.

4.5. North and East Interfaces

The North interface is the OpenStack API. We use a richer semantics that let users specify context information. This information is not directly used by OpenStack, which does not include any specific logic, but is exposed to external management software (e.g., QoS/Consolidation algorithms) via the East interface.

Our implementation exploits the plain OpenStack interface to share context based on the model described in Section 3.2. The specific implementation of the abstract model includes the following information:

- *active/paused* state. According to OpenStack documentation, VMs can be in different steady states, roughly corresponding to ACPI power states. We expect that users make use of the PAUSED state to indicate those VMs that are not currently used (*green* label, i.e., spare components for horizontal scaling or backup), hence implicitly requesting (temporary) low service levels;
- *bandwidth* requirements. These are properties in the form $\langle key, value \rangle$ associated to each VMs and stored in the metadata server, which extend QoS constraints already present in OpenStack API (Currently, QoS support in OpenStack is very limited, and only covers a few configurations in the hypervisor, but none in the physical network);
- *service level*. This is also encoded by a property in metadata server and is representative of the degree of acceptable overprovisioning (*yellow/red* label).

The usage of virtual power states (active/paused) better matches the *green* class to its meaning, facilitates the detection of state transitions, and allows smoother transitions of power states of the underlying hardware.

The East interface is a collection of control and management APIs to exploit the unique features of this kind of infrastructure. From top to bottom on the right side of Figure 1, we find the cloud API, the power management interface, and the network API.

The *cloud API* is the standard OpenStack interface, used in admin mode to retrieve information about the virtualization system (running VMs, state, tenant networks, VLAN ids, etc.) and to trigger migration of VMs. It is also used to detach hypervisors from the OpenStack controller before putting them to sleep, in order to avoid any unreachability error.

The *power management* interface is used to:

- retrieve information about current power consumption;
- build power profiles for servers and network devices by correlating power consumption with different load conditions (CPU usage);
- change the power state of servers and network devices.

The first two operations rely on the monitoring framework described in Section 4.4, whereas the last one uses the GAL for network devices, a custom interface for compute nodes, and the Wake-on-Lan protocol for all devices (see Sections 4.1 and 4.3).

Finally, the *network API* allows programmatic configuration of the communication infrastructure. We use RESTCONF [27], available as an OpenDayLight feature, which provides high-level network abstraction through the Yang model. This interface can be used to set flows in the underlying data path, according to the specific strategies computed by management algorithms.

5. Evaluation and Numerical Results

5.1. Experimental Setup

We deployed a working testbed as depicted in Figure 2. It is a small testbed, but it is rather representative of edge installations, which are not expected to have the same size as modern data centers. It is composed of the hardware and virtual components shown in Table 1.

Table 1. Hardware used in the experimental testbed.

Role	ID	Type	CPU			RAM	Disk	Net
			Model	*Speed*	*Cores*			
OS compute node	C1	Phy	Core i7-6770HQ	2.66 GHz	4	32 GB	256 GB	1 × 1 GB
	C2	Phy	Core i7-6770HQ	2.66 GHz	4	32 GB	256 GB	1 × 1 GB
	C3	Phy	Core i7-6770HQ	2.66 GHz	4	32 GB	256 GB	1 × 1 GB
OS network node	NN	VM	kvm64	2.30 GHz	2	4 GB	32 GB	1 × 1 GB
OS storage node	SN	Phy	Core i5-750	2.67 GHz	4	2 GB	750 GB	1 × 1 GB
Network switch	S1	Phy	Atom N550	1.5 GHz	2	2 GB	500 GB	4 × 1 GB
	S2	Phy	Atom N550	1.5 GHz	2	2 GB	500 GB	2 × 1 GB + 2 × 100 Mb
	S3	Phy	Atom D510	1.66 GHz	2	1 GB	120 GB	2 × 1 GB
	S4	Phy	Atom 230	1.6 GHz	1	4 GB	160 GB	2 × 100 Mb

The installed software is the Newton release for OpenStack, and Carbon for OpenDayLight. We used a mix of bash scripts and simple java programs for triggering our APIs and QoS parameters.

According to similar works in the literature, we selected low-end hardware for our testbed. Indeed, our work leverages innovative technologies (power saving mechanisms and protocols, software-defined networking) that are not yet largely available together in commercial devices for cloud installations.

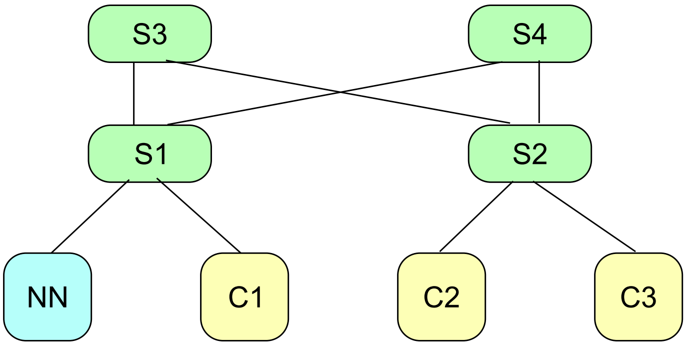


Figure 2. Experimental testbed. NN = Network Node, C = Compute node (server), S = Switch.

5.2. Performance

The application of power-saving mechanisms is expected to impact the Quality of Service (QoS) perceived by the applications, but this aspect is always neglected when the evaluation of consolidation algorithms is carried out by simulations. Indeed, the migration of VMs interrupts the service, and also requires change in the network configuration, to re-route packets. As a first result, we extend the measurements already reported in our previous work [5].

5.2.1. Forwarding Paths

A distinctive characteristic of our framework is the integration with SDN protocols (i.e., OpenFlow), which allows optimal bandwidth usage through the configuration of forwarding paths between VMs ('Explicit path'). When VMs are moved to a different server, the underlying forwarding path has to be updated, and this may result in packet losses, jitter variations, and broken connectivity. To understand the potential impact of network re-configuration, we generated a UDP traffic stream from a VM hosted on C2 to a VM hosted on C1. The stream was generated with *iperf* (https://iperf.fr/), a largely used open-source tool for testing TCP, UDP, and SCTP; the usage of UDP packets enables for measuring performance without the interference of any congestion avoidance and error recovery procedure.

We chose to change the configuration of the network path without migrating VMs, in order to only capture the effects due to OpenFlow operation. The path is changed from the sequence $\langle C1, S1, S3, S2, C2\rangle$ to $\langle C1, S1, S4, S2, C2\rangle$. For comparison, we also consider the plain behavior of the l2switch feature in OpenDayLight ('Pure flooding'), which is used in our framework to provide full connectivity among all nodes.

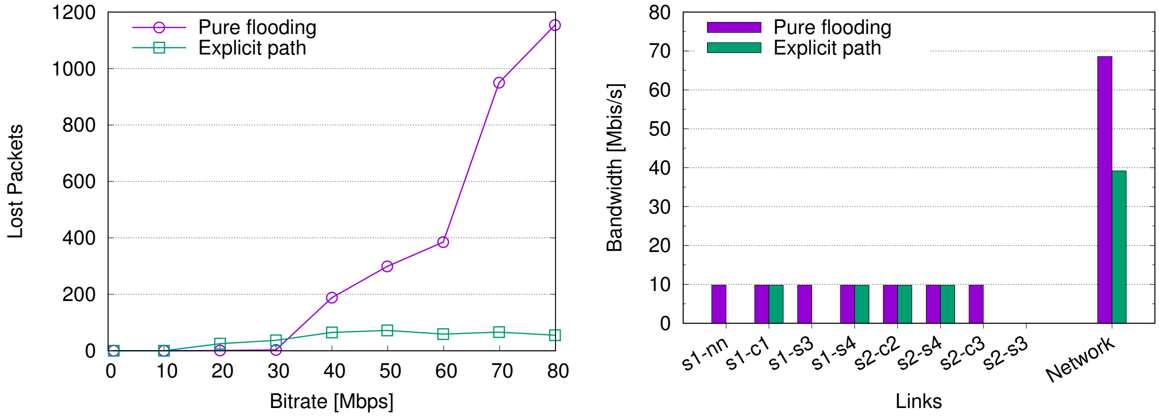
Figure 3a shows that there is no loss with low bit rate (below 1 Mbps). For moderate rates (10–30 Mbps), the l2switch flooding performs better, since there is no need for re-configuration. However, with larger rates, the number of lost packets remains quite constant in case of configured paths, while it rapidly raises for flooding. This happens because losses occur for a short time during the path re-configuration, which is fixed independently of the stream rate, while flooding creates wide network congestion (see Figure 3b).

5.2.2. Live Migration

Live migration is an essential feature for any consolidation algorithm because it moves VMs between different servers with minimal service disruption. However, critical applications may be sensible to service discontinuity, so it is important to evaluate this delay.

To assess the impact of live migration, we measured how the execution time increases when such an event occurs. We considered a transcoding application, which has a deterministic behavior and generates high computing load. Table 2 lists the main parameters for the transcoding application.

The migration process in our experimental setup is controlled by a KVM/QEMU hypervisor. There are several parameters that can be set in the OpenStack configuration files to tune this process; we found the most relevant for performance is the 'live migration downtime'. According to the documentation, live migration copies the instance's memory from the source to the destination hypervisor without stopping the instance. It works in an iterative way, by copying over and over again pages that are written by the instance in the meantime (marked as *dirty* pages). To avoid looping forever, the instance is paused for a short time periodically, so that the remaining few pages can be copied without interference from memory writes. This interval is proportional to the parameter aforementioned; if the memory copy cannot be completed within this internal, the instance is resumed and the process repeats from beginning. The procedure increments the downtime interval linearly at successive iterations until the maximum permitted value (i.e., the live migration downtime parameter) is reached; if a clean copy of the memory is not completed, the migration fails.



(**a**) (**b**)

Figure 3. Comparison of network performance for flooding (l2switch behavior) and explicit path configuration. (**a**) packet loss; (**b**) bandwidth usage.

Table 2. Relevant parameters of the transcoding application.

Parameter	Original Value	Final Value
Video length	ca 22 m 52 s	Same
Transcoding tool	ffmpeg 3.4.1	N/A
Video stream	h264 (native)	mpeg4 (native)
Audio stream	aac (native)	mp3 (libmp3lame)
Container format	mp4	avi
Transcoding time (w/o migration)	N/A	ca 4 m 6 s

In a nutshell, a live migration process periodically interrupts the execution with linearly increasing intervals. This raises the suspect that, overall, smaller downtime values might lead to longer service disruption and unavailability than larger values, even if that looks counterintuitive. In addition, network bandwidth is used intensively during the migration, hence multiple failed attempts also turn into high overhead.

The above impression is partially confirmed by analyzing the total processing time and migration time for video transcoding (see Figure 4). We measured a smaller (almost negligible) increase in the service processing time while using the two larger downtime values. We also see the great reduction in total migration time when using such longer downtime values. From this evaluation, we conclude that setting longer live migration downtime values is often beneficial both from service and migration performances, hence we fixed these parameters to 2000 ms in all successive trials.

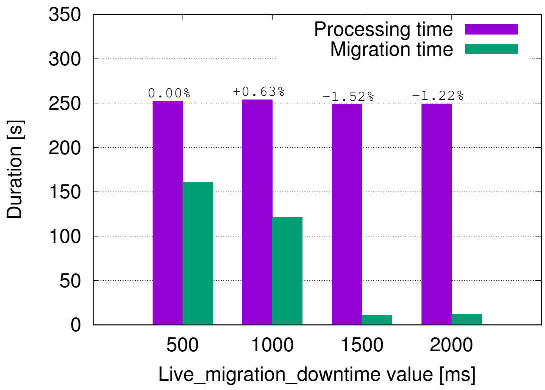


Figure 4. Duration of service processing and migration, in case a VM is migrated between two servers.

5.2.3. Availability

Every power-saving mechanism entails reduced capability (e.g., lower frequency, shut down components), hence some time elapses before returning to full operation. In our case, paused VMs may be hosted on a sleeping server. The time to get them ready for use is some hundred milliseconds for hardware wake-up, plus additional time for the operating system to carry out resume operations; in total, this sums up to a few seconds.

The above value is some order or magnitude smaller than provisioning a VM from scratch. In addition, the main concept behind our approach is that users voluntarily put their VMs in the paused state, so they are aware of the time to get them active again. Based on these considerations, we think that more precise measurements are not necessary at this stage.

5.3. Energy Efficiency: Performance vs. Power Consumption

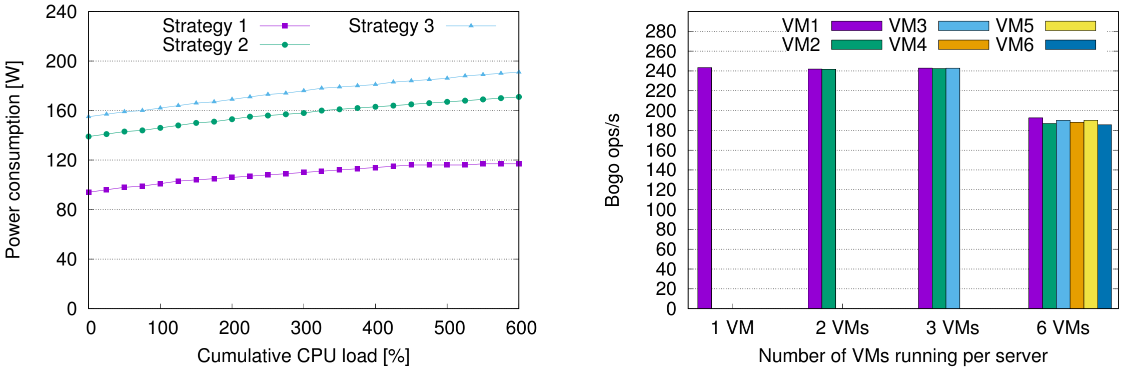
Given the antithetical goals of energy efficiency and QoS, it is natural to wonder how they are balanced in our paradigm. Indeed, the two aspects are often investigated separately, and their correlation is not studied. To this aim, we analyze how consolidation affects the overall efficiency, given by the ratio between performance and power consumption. We suppose having six VMs, and we select three different placement strategies. Each placement corresponds to more or less aggressive consolidation, by using less of more servers, respectively. The placement of VMs for each strategy is described in Table 3. In this case, we perform the evaluation in static conditions for consolidations, i.e., the placement does not change during each experiment.

Table 3. Allocation of VMs for different consolidation strategies.

Strategy	Compute1	Compute2	Compute3
1. Max saving	All VMs (VM1–6)	–	–
2. Balanced	3 VMs (VM1, 3, 5)	3 VMs (VM2, 4, 6)	–
3. Max performance	2 VMs (VM1, 4)	2 VMs (VM2, 5)	2 VMs (VM3, 6)

We evaluated with real data how power consumption and performance change for each strategy. We used the stress-ng tool (http://kernel.ubuntu.com/~cking/stress-ng/) to incrementally increase the CPU load for each VM. We consider 100% as full utilization of one VM, so the total load for six VMs equals 600%; the incremental step is fixed at 25%. We increment the load for each VM in numerical order, i.e., first, we increase the load for VM1 and, when it is full (100%), we start with VM2, and so on until the last VM (VM6) is fully loaded.

We show power consumption and performance for the three strategies in Figure 5. We see that a large power consumption gap is present between Strategy 1 and Strategy 2 (around 40% increase), while the difference is limited between Strategy 2 and Strategy 3 (Figure 5a). This is obvious if we consider the network topology (Figure 2): by placing VMs on Compute2, we need to wake up two additional switches (S2 and S3), which are not necessary for Strategy 1, whereas no additional network devices are necessary when switching from Strategy 2 to Strategy 3.



(**a**) (**b**)

Figure 5. Comparison of power consumption and performance for the three consolidation strategies; (**a**) power consumption; (**b**) performance.

Strategy 1 achieves larger energy saving, but we must also take performance into account for fair comparison. To this aim, we consider the number of operations per seconds, as reported by stress-ng metrics for full-load experiments (i.e., all VMs running at 100% CPU). Though such metric is not a reliable performance indicator and has not been designed to be used for scientifically accurate benchmarking, it is enough to get a notional rough comparison of performance between different set-ups. We note that performance is almost constant while running one, two, or three VMs concurrently on the same hypervisor, but it significantly degrades for six VMs (Figure 5b). This is not surprising, since each processor on the compute nodes has four cores with hyper-threading; though they appear as eight vCPUs to the OS, hyper-threading is not equivalent to having additional real cores.

For better evaluation, we compare in Figure 6 the variation of three relevant indexes for the different strategies, arbitrarily assuming Strategy 1 as base reference (100%): performance, power, and overall efficiency. We consider total operations per seconds as performance index, the power consumption of the whole infrastructure as power index, and the operations over power consumption ratio as efficiency index. We take into consideration three levels of CPU utilization for VMs: 100% (full load), 75% (high load), and 50% (medium load).

Figure 6 shows that, for full-load utilization, Strategies 2 and 3 have almost identical performance (+28% with respect to Strategy 1), but different power profiles (+35% and +44% increase). Strategy 1 is undoubtedly the most efficient solution (+5%, +11% than other Strategies), though this comes at the cost of worse performance. We can say that Strategy 2 is better than Strategy 3, since we have higher efficiency and lower power consumption with the same performance level. Instead, comparison between Strategy 1 and Strategy 2 is not straightforward because it depends on the overall optimization objectives (which may lean towards performance or energy saving). In general, Strategy 1 is preferable, since the energy saving is larger than the performance penalty, hence efficiency is higher with respect to the other scenarios.

When VMs need less CPU time (i.e., lower utilization level), there is more room for overcommitment. Indeed, the performance increment is limited to 15% with 75% CPU load and negligible with 50% CPU load. Since the power increment is comparable for all CPU loads, this turns into greater loss of efficiency from Strategy 1 to Strategy 3.

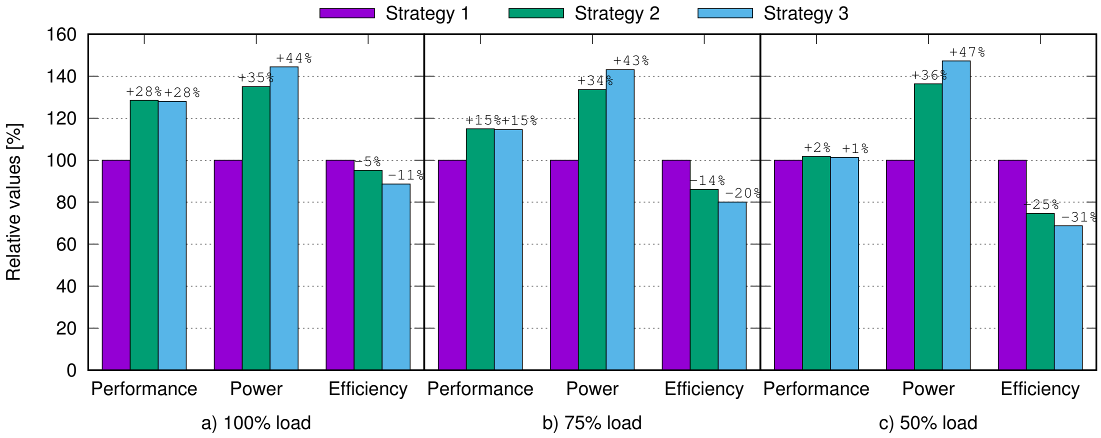


Figure 6. Comparison of performance, power, and overall efficiency for the three consolidation strategies, under different utilization factors.

We conclude by remarking that more aggressive consolidation (Strategy 1) is the optimal choice when VMs are not 100% loaded. Instead, when VMs are fully loaded, less aggressive consolidation brings larger performance gains, which motivates a slight loss of efficiency. In general, less aggressive consolidation is better suited for latency-sensitive and mission-critical applications, where performance should prevail over power efficiency. This observation should be taken into account while designing consolidation algorithms; the labeling scheme used in our consolidation strategy enables for easily taking into account the different requirements, hence supporting the definition of optimal consolidation plans tailoring the need of different classes of applications.

5.4. Consolidation and Energy Saving

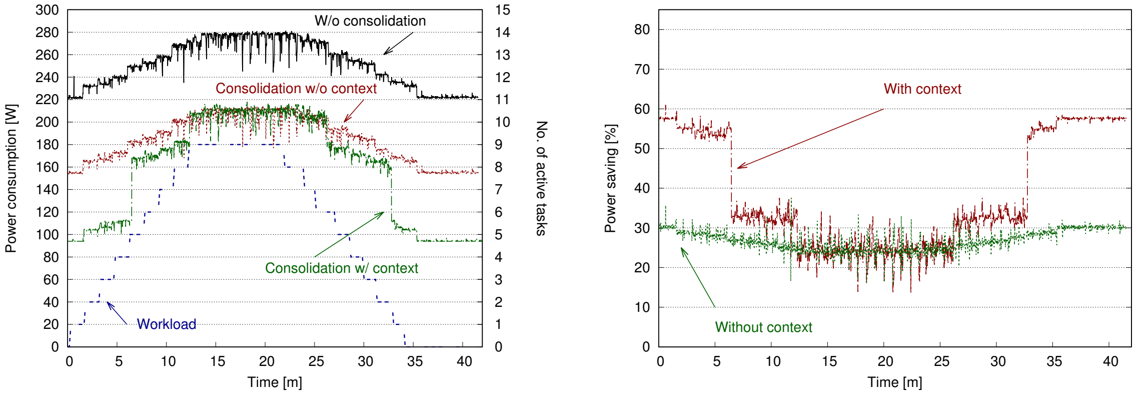
The last part of our work is devoted to evaluation of energy saving with a sample application. The purpose is comparison among three cases: no workload consolidation, workload consolidation without context information, and workload consolidation with context information. To this purpose, we developed a very simple elastic service, made of one control master and several slaves for load balancing. Elastic applications are suitable to tackle variable workload, since they scale according to the current computation need. In our framework, the master delegates processing tasks to the slaves, and we name it "dispatcher"; the slaves carry out computing tasks in parallel, and are indicated as "workers". The application is a video-transcoding service, which uses as many workers as needed to transcode multiple video files in parallel, one on each worker. Since the main purpose is the evaluation of power saving, we do not consider more complex actions as video queuing.

We allocated one VM for the dispatcher and nine VMs for workers (worker1, worker2, . . . , worker9); the dispatcher is considered a critical service ("red" label), while workers are paused/unpaused depending on the current workload. Each VM has 1 vCPU and 2 GB RAM. For simplicity, we do not overcommit resources, i.e., the maximum number of VM per server is limited to 4 (since each server has four physical cores).

We also developed a very simple energy management and consolidation strategy, which makes use of the Eastbound interface. It periodically checks the status of VMs (active vs. paused), and performs consolidation, by clustering VMs on the smallest number of servers, minimizing the overall power consumption of computing servers and network switches. In addition, it puts idle servers in the sleep state, and wakes them up when a request for 'unpause' is received.

The placement strategy for consolidation is a very simple heuristics, which is not worth discussing in detail. The same algorithm is applied when context information is available and when it is not. In the last case, obviously, the status is not considered in the placement decision, and only servers that do not host any VM can be put to sleep. We deliberately did not use advanced and complex algorithms for evaluation, for more fair comparison between the two cases.

We generated a sequence of nine tasks with the same processing time (around 21 min), every 90 s. This leads to a bell-shaped utilization curve. Figure 7a shows the measured power consumption in the three different scenarios: (i) without consolidation; (ii) with a consolidation procedure that does not take into consideration context information; and (iii) with our consolidation scheme that uses context information.



(**a**) (**b**)

Figure 7. Comparison between consolidation without and with context information. (**a**) power consumption; (**b**) power saving.

The utilization shape is reflected in power consumption, which increases when more tasks are present (see Figure 7a). Without context information, all VMs are treated equally by the consolidation procedure, independently of their actual utilization and role. This basically prevents servers from sleeping (since the number of VMs is quite large with respect to the number of servers, and there is no overcommitment); only one redundant switch can be put to sleep. Instead, when our simple orchestration policy changes the state of unused VMs to "PAUSED", the consolidation algorithm takes them apart and some servers and switches can sleep, hence leading to lower energy consumption in the presence of low workload.

Figure 7b shows power saving of the two consolidation strategies with respect to the scenario with no consolidation. We note that, in both cases, power saving decreases with higher processing load, as expected.

Figure 8 shows that, using context information, we can achieve a better linear relationship between server utilization and power consumption. Indeed, the linearity is more evident in lower power consumption corresponding to scarce utilization. The overall curve has a more stepwise shape with respect to other scenarios, and this can be ascribed to the small number of devices in the testbed. When more devices are used, together with dynamic voltage/frequency scaling mechanisms, the stepwise behavior becomes negligible.

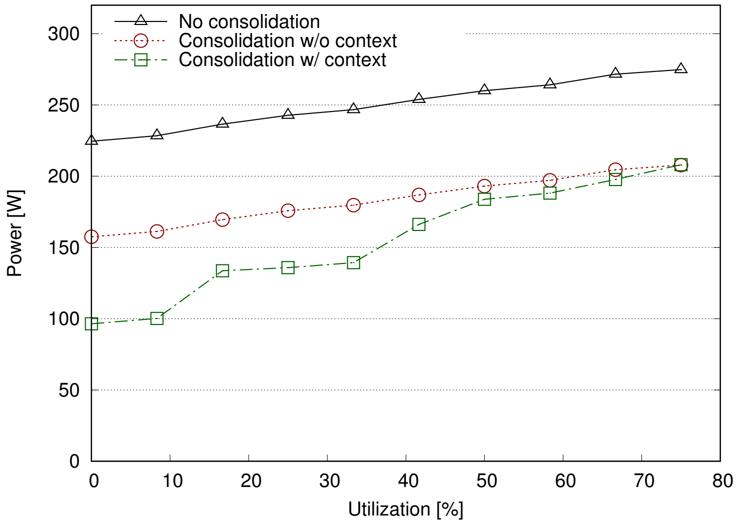


Figure 8. Power consumption vs. server utilization.

6. Conclusions

A lot of effort has been devoted in recent years to designing consolidation strategies that maximize the effectiveness of power-saving mechanisms. These approaches do not take into account the actual context, i.e., whether virtual resources are really used or just provisioned for other purposes (including service migration, availability and resilience). Orchestration paradigms can be effectively used to reflect the current role of each virtual resource in proper infrastructure parameters; for instance, we inserted metadata and applied virtual power states in a very similar way to physical hardware. This allows more awareness in the consolidation strategy and brings better opportunities to selectively apply aggressive power-saving mechanisms.

Through orchestration, users are left the responsibility to accept light penalties in responsiveness and availability, and their willingness to take part in such mechanisms may easily be fostered by suitable pricing schemes. This is the main strength behind the overall approach, which can effectively balance power consumption with QoS in edge computing.

Author Contributions: Conceptualization, M.R.; Funding acquisition, R.B.; Investigation, A.C. and M.R.; Methodology, M.R.; Project administration, R.B.; Software, A.C. and G.R.; Visualization, M.R.; Writing—Review & Editing, M.R.

Funding: This work was supported in part by the European Commission under the projects ARCADIA (contract No. 645372) and MATILDA (contract No. 761898).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Soldani, D.; Manzalini, A. On the 5G Operating System for a True Digital Society. *IEEE Veh. Technol. Mag.* **2015**, *10*, 32–42. [CrossRef]
2. Kennedy, D.; Bourse, D.; Mohr, W.; Bedo, J.-S.; Herzog, U. *5G Empowering Vertical Industries*; Whitepaper from the 5G-PPP, ERTICO, EFFRA, EUTC, NEM, CONTINUA and Networld2020 ETP, 2016. Available online: https://5g-ppp.eu/wp-content/uploads/2016/02/BROCHURE_5PPP_BAT2_PL.pdf (accessed on 20 June 2018).
3. Bolla, R.; Khan, R.; Repetto, M. Assessing the Potential for Saving Energy by Impersonating Idle Networked Devices. *IEEE J. Sel. Areas Commun.* **2016**, *34*, 1676–1689. [CrossRef]
4. Carella, G.A.; Pauls, M.; Magedanz, T.; Cilloni, M.; Bellavista, P.; Foschini, L. Prototyping nfv-based multi-access edge computing in 5G ready networks with open baton. In Proceedings of the IEEE Conference on Network Softwarization (NetSoft), Bologna, Italy, 3–7 July 2017.

5. Carrega, A.; Portomauro, G.; Repetto, M.; Robino, G. OpenStack extensions for QoS and energy efficiency in edge computing. In Proceedings of the 3rd IEEE International Conference on Fog and Edge Mobile Computing (FMEC 2018), Barcelona, Spain, 23–26 April 2018.
6. Barroso, L.A.; Hölzle, U. The case for energy-proportional computing. *Computer* **2007**, *40*, 33–37. [CrossRef]
7. Shirayanagi, H.; Yamada, H.; Kono, K. Honeyguide: A VM migration-aware network topology for saving energy consumption in data center networks. In Proceedings of the IEEE Symposium on Computers and Communications (ISCC), Cappadocia, Turkey, 1–4 July 2012; pp. 460–467.
8. Li, B.; Li, J.; Huai, J.; Wo, T.; Li, Q.; Zhong, L. EnaCloud: An Energy-Saving Application Live Placement Approach for Cloud Computing Environments. In Proceedings of the IEEE International Conference on Cloud Computing (CLOUD '09), Bangalore, India, 21–25 September 2009; pp. 17–24.
9. Buyya, R.; Beloglazov, A.; Abawajy, J. Energy-Efficient Management of Data Center Resources for Cloud Computing: A Vision, Architectural Elements, and Open Challenges. In Proceedings of the 2010 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2010), Las Vegas, NV, USA, 12–15 July 2010.
10. Heller, B.; Seetharaman, S.; Mahadevan, P.; Yiakoumis, Y.; Sharma, P.; Banerjee, S.; McKeown, N. ElasticTree: Saving Energy in Data Center Networks. In Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation, San Jose, CA, USA, 28–30 April 2010; USENIX Association: Berkeley, CA, USA, 2010; p. 17.
11. Voorsluys, W.; Broberg, J.; Venugopal, S.; Buyya, R. Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation. In Proceedings of the 1st International Conference on Cloud Computing (CloudCom '09), Beijing, China, 1–4 December 2009; pp. 254–265.
12. Beloglazov, A.; Buyya, R. OpenStack Neat: A framework for dynamic and energy-efficient consolidation of virtual machines in OpenStack clouds. *Concurr. Comput. Pract. Exp.* **2015**, *27*, 1310–1333. [CrossRef]
13. Rossigneux, F.; Gelas, J.P.; Lefèvre, L.; de Asunção, M.D. A Generic and Extensible Framework for Monitoring Energy Consumption of OpenStack Clouds. In Proceedings of the 2014 IEEE Fourth International Conference on Big Data and Cloud Computing, Sydney, NSW, Australia, 3–5 December 2014.
14. Carrega, A.; Repetto, M. Energy-Aware Consolidation Scheme for Data Center Cloud Applications. In Proceedings of the 2017 29th International Teletraffic Congress (ITC 29), Genoa, Italy, 4–8 September 2017.
15. Cima, V.; Grazioli, B.; Murphy, S.; Bohnert, T. Adding energy efficiency to Openstack. In Proceedings of the Sustainable Internet and ICT for Sustainability (SustainIT), Madrid, Spain, 14–15 April; pp. 1–8.
16. Shehabi, A.; Smith, S.; Sartor, D.; Brown, R.; Herrlin, M.; Koomey, J.; Masanet, E.; Horner, N.; Azevedo, I.; Lintner, W. *United States Data Center Energy Usage Report*; Technical Report LBNL-1005775; Ernest Orlando Lawrence Berkeley National Laboratory: San Jose, CA, USA, 2016.
17. Avgerinou, M.; Bertoldi, P.; Castellazzi, L. Trends in Data Centre Energy Consumption under the European Code of Conduct for Data Centre Energy Efficiency. *Energies* **2017**, *10*, 1470. [CrossRef]
18. Brady, G.A.; Kapur, N.; Summers, J.L.; Thompson, H.M. A case study and critical assessment in calculating power usage effectiveness for a data centre. *Energy Convers. Manag.* **2013**, *76*, 155–161. [CrossRef]
19. Yuventi, J.; Mehdizadeh, R. A critical analysis of Power Usage Effectiveness and its use in communicating data center energy consumption. *Energy Build.* **2013**, *64*, 90–94. [CrossRef]
20. Wang, C.; Nasiriani, N.; Kesidis, G.; Urgaonkar, B.; Wang, Q.; Chen, L.Y.; Gupta, A.; Birke, R. Recouping Energy Costs From Cloud Tenants: Tenant Demand Response Aware Pricing Design. In Proceedings of the 2015 ACM Sixth International Conference on Future Energy Systems (e-Energy '15), Bangalore, India, 14–17 July 2015; pp. 141–150.
21. *Topology and Orchestration Specification for Cloud Applications*, version 1.0; OASIS Standard, 2013. Available online: http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.pdf (accessed on 20 June 2018).
22. Network Functions Virtualisation (NFV); Management and Orchestration. *ETSI GS NFV-MAN 001 V1.1.1*; European Telecommunications Standards Institute: Sophia Antipolis, France, 2014.
23. DTMF. *Power State Management Profile*, version: 2.0.0; Specification DSP1027; 2009. Available online: https://www.dmtf.org/sites/default/files/standards/documents/DSP1027_2.0.0.pdf (accessed on 20 June 2018).
24. Hass, J. *IPMI CIM Mapping Guideline*; Document Revision 0.60; 2006. Available online: https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/cim-mapping-guideline-.6.pdf (accessed on 20 June 2018).

25. *Green Abstraction Layer (GAL); Power Management Capabilities of The Future Energy Telecommunication Fixed Network Nodes*, version 1.1.1; ETSI ES 203 237; European Telecommunications Standards Institute: Sophia Antipolis, France, 2014.
26. Kanapram, D.; Lamanna, G.; Repetto, M. Exploring the trade-off between performance and energy consumption in cloud infrastructures. In Proceedings of the 2nd IEEE International Conference on Fog and Edge Mobile Computing (FMEC 2017), Valencia, Spain, 8–11 May 2017; pp. 121–126.
27. Bierman, A.; Bjorklund, M.; Watsen, K. *RESTCONF Protocol*; RFC 8040; Internet Engineering Task Force, 2017. Available online: https://tools.ietf.org/html/rfc804 (accessed on 20 June 2018).