Smart management of renewable energy in clouds : from infrastructure to application

Md Sabbir Hasan

Thèse

UNIVERSITE
BRETAGNE
LOIRE

THESE INSA Rennes
sous le sceau de l'Université Bretagne Loire
pour obtenir le titre de
DOCTEUR DE L'INSA RENNES
Spécialité : Informatique

présentée par

MD Sabbir Hasan

ECOLE DOCTORALE : *MATISSE*
LABORATOIRE : *IRISA*

Smart Management of Renewable Energy in Clouds: from Infrastructure to Application

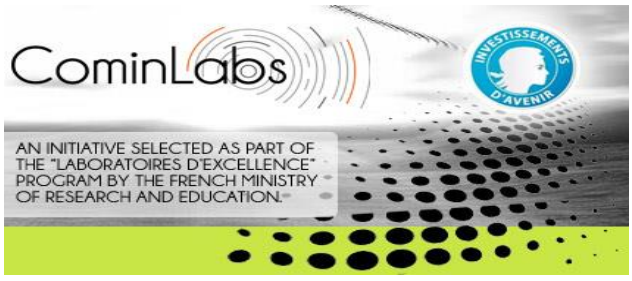
Thèse soutenue le 03.05.2017
devant le jury composé de :

Christine MORIN
DR, INRIA Rennes-Bretagne Atlantique / Présidente
Jean-Marc PIERSON
PU, Université Paul-Sabatier, Toulouse / Rapporteur
Pascal BOUVRY
PU, Université du Luxembourg, Luxembourg / Rapporteur
Eric RUTTEN
CR, INRIA Grenoble Rhône-Alpes / Examinateur
Romain ROUVOY
PU, Université de Lille, Lille / Examinateur
Thomas LEDOUX
MA, IMT Atlantique, Nantes / Co-encadrant de thèse
Jean-Louis PAZAT
PU, INSA Rennes / Directeur de thèse

Smart Management of Renewable Energy in Clouds : from Infrastructure to Application

MD Sabbir Hasan

En partenariat avec



Acknowledgment

This is the last portion of writing that is going to be attached in the manuscript which didn't go through either a peer reviewed process or any major/minor revision! Therefore, empathy might overpower the lucidity of my mind.

Firstly, I would like to extend my profound gratitude to both of my PhD advisors, *Prof. Jean-Louis Pazat* and *Thomas Ledoux* for their continuous guidance, advice, patience throughout my PhD duration. Without their support and direction, this thesis would have not been possible. I would like to give Thomas a special thanks for tolerating my somewhat impatient mind and stubbornness at times and giving me flexible working hours and freedom to explore.

I would like to thank Prof. Jean-Marc Pierson and Prof. Pascal Bouvry for accepting the request to be reviewer of this thesis and for their valuable insights and suggestions to improve the quality of the manuscript. Additionally, I would like to express my sincere gratitude to the Jury members Christine Morin, Eric Rutten and Romain Rouvoy for their valuable suggestions, challenging questions and admiration during the thesis defense to make the occasion much memorable and unfathomable.

Looking back, It's all started in the summer of 2013 in August, when i came back to Bangladesh after spending 2 years at South Korea for my masters degree. I received an email followed by a skype interview and boom! I was selected for doing a PhD at INSA, Rennes. Amidst that period, i was planning to move to Canada to start my PhD. But i felt the temptation to experience better weather, food, diverse culture etc. hence my heart lean towards to Western Europe.

My journey became a lot easier for having some amazing colleagues like Simon, Yousri and Frederico with whom i had scientific discussion, written articles, passed sleepless nights over email and Slack in the eve of some conference deadline. I would also like to thank other members of ASCOLA and MYRIADS team members for their help and presence throughout my PhD journey.

To my friends at *Ecole des Mines de Nantes* for always having weird, hodgepodge and heated discussions on food, weather, movies, songs, travel, politics, sports etc. during long post-lunch-coffee-breaks. Sometimes these discussion were extended to weekend parties. Thank you all of my friends for being fun, nerdy, inspiring and sometimes irritating! I would also like to thank my friends back in home and abroad and to them who stayed

briefly in my life but encouraged me to do better things.

I want to remember my parents, brother and other family members for their immense contribution from day 1 at school to the end of my academic degree career and thank them wholeheartedly.

Finally, I would like to give a big thank to my wife, Tama for always being there in my ups and downs and sometimes enduring my irrational minds at times and late working hours at night.

MD Sabbir Hasan, June 2, 2017

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As a direct consequence of the increasing popularity of Internet and Cloud Computing services, small-scale to large-scale data centers are rapidly growing. In 2007, data centers in Western Europe consumed a whopping 56 terawatt-hours (TWh) of power per year. According to the EU, this figure is likely to almost double to 104TWh by 2020[1]. In the same year, Gartner reported, ICT industry accounts for 2% of global carbon emissions. This high carbon emissions is the result of producing electricity from fossil fuels or coals. Although, France generates 75% of the electricity by nuclear plants which emits relatively lower carbon, the amount of carbon footprint is nowhere near to zero. Apart from that, the energy costs due to power draw and distribution of a data center accounts to 15% of total cost of ownership (TCO).

One of the main reason of the energy consumption growth is that more and more service providers are shifting their application as well as their IT workloads to the Cloud. In 2016, Rightscale[2] reported, the adoption of Cloud services have increased by 13-14% compared to 2015. The foremost reason for moving to the Cloud is to decrease the IT-related costs, complexities and being operational without investing heavily on or maintaining their own computing infrastructure. Additionally, Software-as-a-Service (SaaS) providers or application owners seek to guarantee certain level of performance and availability of their services a.k.a interactive Cloud applications without any disruption to the end users. To guarantee the QoS terms, Cloud applications should be always running and responsive irrespective to user traffic suggesting, Infrastruture-as-a-Service (IaaS) providers need to over-provision the resources but SaaS providers are willing to pay only what they consume. In contrast, IaaS providers aim at running lesser physical machines not only to reduce TCO but also to reduce energy consumption and their associated footprint. While TCO can be decreased via under-provisioning of resources, may degrade QoS properties of the hosted

[1] European Commission, Code of Conduct on Data centers Energy Efficiency- Version 1.0, October 30, 2008

[2] http://www.rightscale.com/blog/cloud-industry-insights/cloud-computing-trends-2016-state-cloud-survey

application. Therefore, the conflicting goals in different service layers are managed by Service Level Agreement (SLA), that is a formal description of temporal, performance and economical constraints between SaaS and IaaS Provider.

1.1 Problem Statement

While the proliferation of Cloud services which reside in data centers is greatly impacting our society, the *greenness* of their nature remains questionable. Greenness can be derived from energy efficient techniques and quality of energy consumed in those data centers. In response, existing researches have focused more on reducing energy consumption by designing/implementing server consolidation [BAB12] [HH13], hardware with better power and performance trade-offs [VAN08], workload migration [BJT+09] and software techniques for energy aware scheduling [KMAHR14], etc. Although these efforts are necessary, the goal of alleviating carbon footprint is far form the expectation. According to a recent report[3], data center electricity consumption increased by about 4% from 2010-2014, a large shift from the 24% increase estimated from 2005-2010. However, energy usage is expected to increase continuously with the same rate of 4% for 2014-2020 like past five years. Therefore, aforementioned data indicates that, energy efficiency alone is not going to reduce the carbon footprint since energy consumption will continue to grow. Rather, explicit or implicit integration of renewable energy to the data center can be a complementary/integral measure along with state-of-the-art energy efficiency technique to reduce further carbon footprint.

Problem 1. Most of today's data center is only connected to electrical grid where electricity is produced through burning coal and natural gas, which are carbon-intensive approaches to energy production. Therefore, if the electrical grid is not environmental friendly in terms of offsetting and neutralizing net-zero carbon emission or brown energy, large consumers like data centers need to look towards alternative green measures *i.e.*, on-site and off-site green plants, green products etc. *The considerable research challenge using green energy sources in data center are that, they are intermittent by nature, thus always not available.* Apart from that, most of the research articles are unaware of the cost analysis of the variant nature of green energy prices, hence consider green energy incurs $0 [WZL14][APKSG14]. Therefore, exploring different renewable energy integration options and prices can lead to an efficient energy management policy in data center to tackle the intermittent nature of green energy. Once the problem is resolved, IaaS providers can propose green computing services to the SaaS consumers or to end clients.

Problem 2. Applications which are hosted in data center can be roughly classified as *Batch* and *Interactive* applications. While former can be characterized as delay-tolerant,

[3] https://eta.lbl.gov/publications/united-states-data-center-energy

the latter can be very sensitive to delays, otherwise QoS can be heavily impacted. Like any management technique, an efficient energy management can subject to be constrained of not having enough green energy, be it from onsite plant or from wholesale energy market *i.e.,* in case of natural disaster, cloudy days, high energy price, technical plant problem, competitors with better purchasing power and needs etc. To this, several work has been proposed and practiced on how to schedule or run Batch jobs when green energy is available so to consume more green energy and not waste any of it. This results in reducing brown energies in data center, which impact positively to the carbon footprint reduction goal. On the other hand, Interactive applications workload/traffic response can not be delayed and that rules out scheduling of any tasks. Therefore, the research question is: *how to make Interactive SaaS application adaptive to green energy availability while traditional QoS properties can be at satisfactory level so to lower carbon footprint?*. Moreover, green energy adaptivity in interactive cloud applications has not yet been addressed in existing research. Since societal and environmental concerns have been prompting green energy initiatives, it's high time to consider greenness of energy metric as an essential attribute along with traditional QoS.

Problem 3. While IaaS layer allows to dynamically adjusting the provision of physical resources according to Platform-as-a-Service (PaaS) needs for optimizing energy efficiency of the data center, reducing carbon footprint is still underachieved. Integrating different renewable energy options at data center level and adoption of Autonomic Computing (AC) at SaaS layer for greenness, responsiveness and autonomy in front of environmental changes could be a feasible solution for bettering the *Cloud eco-system*. Additionally, at the SaaS layer, AC can enable applications to react to a highly variable workload and presence of renewable energy by dynamically adjusting the amount of resources in order to keep the QoS for the end users. However, problems may occur since those self-managed systems are related in some way (*e.g.,* applications depend on services provided by a cloud infrastructure). Dynamic adaptation of an application can lead to more/less resource requirements that can be provisioned or de-provisioned on the fly. Therefore, decisions taken in isolation at given layer may mismatch the resource requirements by the application, that can negatively impact QoS and energy consumption reduction goal.

Therefore, by creating green energy awareness in the interactive application and smartly adapting in a self-adaptive manner to the changing condition can be the only way to reduce further carbon footprint.

1.2 Contribution

Therefore, in this thesis, to tackle the problems discussed at Section 1.1, we follow a bottom to top (*e.g.,* Infrastructure to application) approach. Our contribution in this thesis are as

follows:

- We first seek to investigate the options and challenges to integrate different renewable energy sources in a realistic way due to it's intermittent nature to better manage energy in the data center. To tackle the problem, we first propose a *Cloud energy broker*, which can adjust the availability and price combination to buy Green energy dynamically from the energy market in advance to make a data center partially green. Data center operator can decide until what percentage they want to consume green energy and plan in advance. Later we introduce the concept of *Virtualization of Green Energy*, that suggests, energy can be virtually green for a specific period of time if the abundance of green energy is available aperiodically in shorter time intervals along with the deficit of green energy in rest of the time frame. Therefore, the virtualization concept can increase greenness of energy, rather increasing the actual amount of green energy. Traditionally, abundant energy is stored in energy storage to use opportunistically when production of green energy is scarce. But for a small scale data center (10-50 servers), using energy storage might not be an efficient solution. With the help of virtualization of green energy, we propose a *virtual battery* which can store and discharge energy virtually having a maximum capacity. Furthermore, we propose *GreenSLA* (One of the first of it's kind based on green energy) by the taking advantage of Virtualization of Green Energy concept. Figure 1.1 illustrates the overview of proposed solution and (1) indicates the position of the aforementioned contribution.

  Even with the virtualization of green energy, which is a coarse-grained concept, three events can occur named as *insufficient, ideal, surplus*. Insufficient event indicates, lack of green energy in data center even though the virtualization concept was adopted. Whereas, surplus event depicts of having more green energy in data center than the requirement, which exceeds the virtual battery capacity.

- We then investigate how we manage Interactive Cloud applications corresponding to the events that is discussed in previous section. Since interactive applications cannot be scheduled in advance, *green energy adaptivity* can only be realized if the application inherits the capability to smartly use the available green energy or to adapt to the three events we mentioned in previous contribution. Firstly, we accomplish this task by finding optional, loosely-coupled and compute intensive components of any application that is not related to core functionality of the service but can provide some extra features. For example, product recommendations at e-commerce application can be seen as extra functionality of a service which is not varily required, but if provided, user experience can be enhanced. Since, these components are resource intensive, secondly we rely on devising and formulating strategies where these components can be activated/deactivated via API calls facing to changing conditions (*e.g.*, workload surge, QoS degradation, shortage/abundance of green energy etc.) in

an autonomic manner at run-time. By reducing resource requirement, not only the energy consumption can be reduced but also targeted QoS can be met and improved if the resource requirement is higher. (2) and (3) at Figure 1.1 illustrates our second contribution. This part of contribution aims at providing insights and strategies needed to make interactive SaaS application green energy aware adaptive.

- While second contribution focuses more on adapting application to changing environment for keeping the targeted QoS with reduced energy consumption, it lacks the capability to utilize the underlying elastic infrastructure. Therefore, our ongoing investigation includes on how to efficiently utilize the elasticity nature of the infrastructure resources when overall resource requirement of an application is higher than the existing underlying infrastructure can handle. Actions like adding/removing resource can be done independently at the infrastructure layer based on their utilization level *i.e.*, cpu usage, memory usage etc. But, every application performs differently from one to another at same cpu utilization level, specially when the resource utilization is medium to high. Therefore, coordinating the decision based on applications resource requirement or performance is the better way to devise scaling strategies. To this, firstly we propose to listen events from application, which is marked by (4) at Figure 1.1 to understand when to trigger scaling decision based on reactive scaling rules. Secondly, we use traditional API such as *scale-in* and *scale-out* to trigger decision based on the strategy we have devised, which is illustrated as (5) at Figure 1.1.

1.3 Outline

This thesis is divided in two parts, which is constructed as follows:

State-of-the-art. This part is consist of Chapter 2 and Chapter 3. Specially in Chapter 2, we present main concepts and definitions of the cloud computing paradigm as well as different classification of Cloud applications based on metrics of interest. Next at Chapter 3, we review the related work around the addressed problem domain and highlights about what is missing in the literature.

Contribution. The second part is comprised of Chapter 4, 5, 6. Chapter 4 investigates the opportunity to exploit the energy market to plan, forecast and purchase energy in advance. Since, any planning or forecasting method are prone to error statistics, Chapter 5 introduces Virtualization of green energy concept to tackle both the forecasting error and intermittency of green energy to propose and revise the notion of GreenSLA. The idea is to propose new class of explicit SLO mentioning the percentage of green energy provided along side with computing services by managing the underlying energy infrastructure

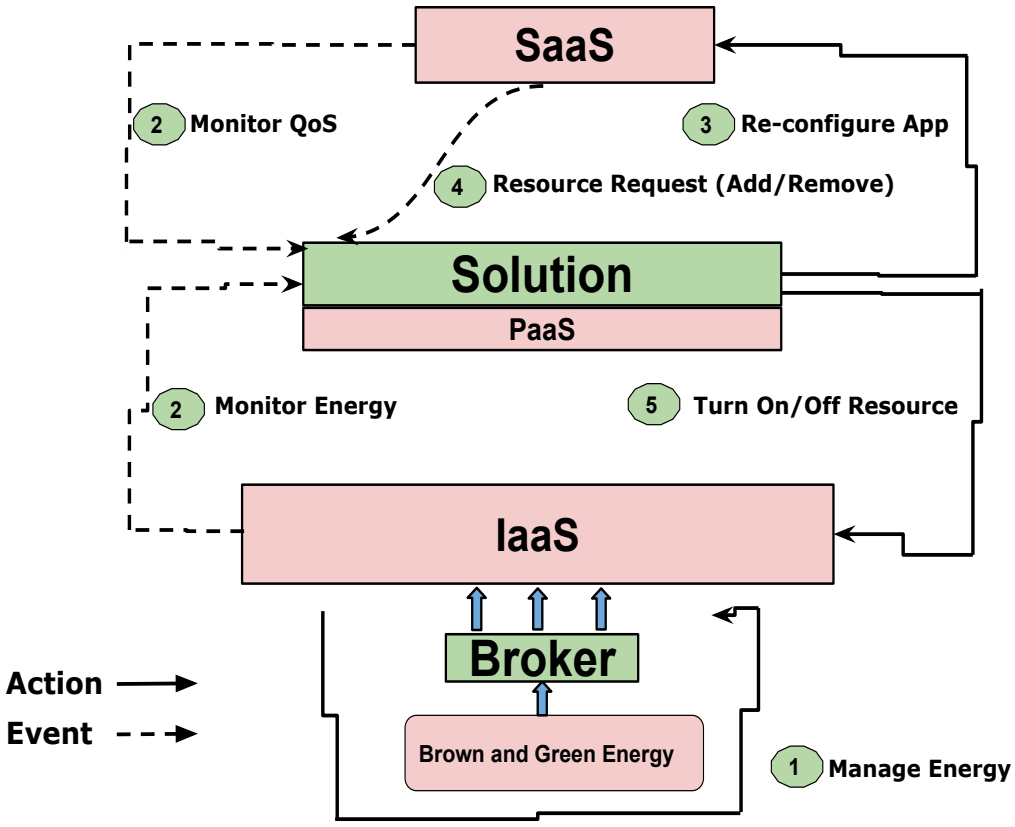


Figure 1.1: Overview of proposed solution

and multi-source energy market. GreenSLA gives the possibility to application owners to host their application in an explicitly expressed green cloud environment having formal contracts. Furthermore, Chapter 6 presents a self-adaptive autoscaler architecture (thanks to autonomic computing) to enable smart usage of energy in an interactive application. The autoscaler inherits the capability of sensing information as events from multiple layer while actions are performed only in application level. Thus, the proposed approach can make an application adaptive by automatically adjusting to changing conditions, while respecting QoS properties.

Chapter 7 presents an ongoing investigation which proposes Green PaaS solution that inherits the capability to adapt both at application and infrastructure level facing to changing conditions. It provides the platform architecture along with some extended application controllers which are able to request of adding/releasing resources to an

proposed infrastructure controller.

Conclusion. Chapter 8 concludes this thesis by revisiting the problem statement, summarizing the contributions, discussing advantages, insights and limitations of our proposed solutions. Later, we discuss some of the possible directions and ideas that could create new future challenges based on the contribution of this thesis.

Part I

State of the art

Chapter 2

Background

This chapter presents very thorough yet concise definitions and explanations of various terms used in the entire thesis. Although Cloud computing domain is widely adopted and well known to the community already, we start the chapter by explaining Cloud computing definitions and its delivery and deployment model. Afterwards, we provide description and some examples of how Service Level Agreement works in the Cloud computing domain. Later, we classify the Cloud application into different categories with their metrics of interest. Lastly, we discuss about energy management problems and opportunities in the context of Cloud computing environment.

2.1 Cloud Computing

What do vast majorities of people think about Cloud? A condensed watery vapour floating in the atmosphere or iCloud[1], Dropbox[2], Google Drive [3] etc. or both of them but from a different perspective? While, the first connotation is true, the latter view represents an unsophisticated way to speculate what Cloud really is in modern computing world. Cloud, which refers to Cloud computing, according to Buyya [BSYV+09]

> *"A Cloud is a type of parallel and distributed system consisting of a collection of interconnected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resource(s) based on service-level agreements established through negotiation between the service provider and consumers."*

Therefore, the above definition is market-oriented computing where Cloud resources are provisoned depending on service level agreement and users are charged based on their usage, just like any other utility *e.g., electricity*. When the definition arrived, Cloud

[1] https://www.icloud.com/
[2] https://www.dropbox.com/
[3] https://www.google.com/drive/

computing was still its infancy, confusing and just a new hype. Later in 2009, Vaquero et al. [VRMCL09] put scrutiny on more than 20 definitions of cloud computing which were in the air and proposed their own definition:

> *"(1) Clouds are a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services); (2) These resources can be dynamically reconfigured to adjust to a variable load (scale), allowing also for an optimum resource utilization; (3) This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure Provider by means of customized SLAs."*

Therefore, the definition suggests that, Clouds are enabled by virtualization having intrinsic capability of scalability and utility model like pay-per-use. If we further dive into the ocean to explore a standardized interpretation, National Institute of Standards and Technology (NIST) [MG11] has given a widely adopted and proper definition:

> *"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."*

That being said, former popular products are just Cloud based storage services, which are enabled by hardware and software virtualization techniques, multicore processors, network virtualization, network overlay, virtual firewall and so on; not the whole paradigm of Cloud Computing. Although, all the definition provide a vague abstraction, we can break down the whole concept into pieces for better understanding and readership in the following subsections.

2.1.1 Cloud service delivery model

Traditionally Cloud computing is composed of three service layers namely: Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS), through which the cloud services are offered. Figure 2.1 shows the service layers of Cloud computing briefly.

SaaS : Principally, SaaS is a software distribution model in which application is deployed and delivered over the internet. It means, the software application is hosted in cloud infrastructure and the consumers have no control over the application and the underlying platform and/or infrastructure. Moreover, software is licensed on a subscription basis and is hosted centrally. While this is a broad generalization, a typical SaaS application needs to have basic characteristics of on-demand self-service, resource pooling and rapid elasticity capabilities and ability to measure the service.

The popular example of SaaS application could be Salesforce.com[4] and LinkedIn[5], where users subscribe themselves and pay a subscription fees for their usage. Google Apps is a perfect example of a SaaS application, where Service Level Agreement (Subscription fees against measured service quality) is available and users have limited degrees of freedom in customization of their apps.

PaaS : PaaS is analogous to SaaS except that, rather than being software delivered over the web, it is a platform for the creation of software, delivered over the web. This platform is a dream for developers since it provides independent runtime and language environment. Besides some of the daunting tasks like deploying, testing and debugging of codes have been made less worrisome, resulting developers only need to think about their codes. In other word, PaaS eliminates the expense and complexity of evaluating, buying, configuring, and managing all the hardware and software needed for custom-built applications. The interesting feature of this layer could be the possibility to monitor both the application and the underlying infrastructure that enables two core characteristics of Cloud computing; ***scalability*** and ***elasticity***. Now, what are these terms? Well, when the traffic to a application increases suddenly, what does exactly happen? Either it cannot admit the user requests or it does not have sufficient resources to serve the request or could be both in general. With the increase of traffic or workload, the ability to increase the resource capacity to the service is termed as scalability, whereas the capability of taking actions dynamically in regards to both increasing and decreasing traffic or workload is elasticity. Some of the popular PaaS service providers are Google App Engine[6], Heroku [7], Mendix[8] etc. Usually PaaS services communicate with SaaS application or services via programmable interface (API).

IaaS : Lastly, IaaS is a way of delivering Cloud computing infrastructure such as servers, storage, network and operating systems in a virtualized manner to build the platform for the creation of software that is delivered over the web. Therefore, virtualization technology is the key enabler of Cloud computing that has revolutionized the whole way of looking at computing in today's world. In simple words, virtualization means to create a virtual version of a device or a resource, be it server, storage device, network or even operating system. Hence, consumers of this layer do not control the physical infrastructure, rather control virtualized resources and deployed application. Unlike SaaS and PaaS, IaaS offers their service in pay-per-use model meaning customer has access to potentially unlimited resources but only pays for what they actually use. The biggest player in the IaaS providing field is Amazon

[4] https://www.salesforce.com/
[5] https://www.linkedin.com/
[6] https://cloud.google.com/appengine/
[7] https://www.heroku.com/
[8] https://www.mendix.com/

AWS[9], which provides Linux based EC2 compute clouds for hourly rates, while other providers like Microsoft Azure[10] and Google compute engine[11] offer 2-3 different operating systems along with their infrastructure. PaaS providers like Heroku and Google App engines are hosted on Amazon EC2 and Google Compute Engines respectively. So, when developers use respective PaaS services, they are restricted to particular IaaS provider's infrastructure.

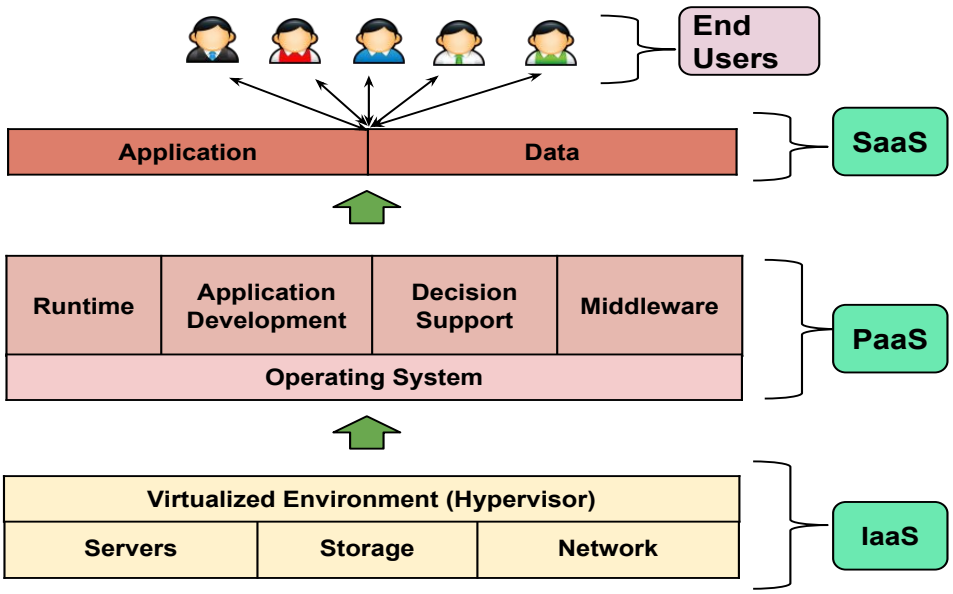


Figure 2.1: Service layers of Cloud computing

2.1.2 Cloud deployment model

Cloud deployment model can be very critical for large to small organizations based on their needs. According to NIST definition of deployment models:

- Private cloud: The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (e.g., business units). It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises.

[9] https://aws.amazon.com/
[10] https://azure.microsoft.com/
[11] https://cloud.google.com/compute/

- Community cloud: The cloud infrastructure is provisioned for exclusive use by a specific community of consumers from organizations that have shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be owned, managed, and operated by one or more of the organizations in the community, a third party, or some combinations of them, and it may exist on or off premises.

- Public cloud: The cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. It exists on the premises of the cloud provider.

- Hybrid cloud: The cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds).

2.2 Service Level Agreement

An explicit agreement of rights and obligation between service consumer and provider involved in service purchase can be formalized as a Service Level Agreement (SLA). Therefore, the obligation and specification changes from one service layer to another depending on the metrics. For example, at the IaaS level, the SLA[12] established by Amazon EC2 and their consumer states that, Amazon provides a monthly uptime percentage of at least 99.95%. In case Amazon can not meet the obligation, the consumer receives service credits based on the agreement. At the PaaS level, Google App Engine provides SLA[13] based on different metrics *i.e.,* Downtime period, Downtime percentage, Error rate for accessing services, etc. In contrast, at the SaaS level, availability of the service is most important and common ground for any service provider. For example, Google Apps provide services mentioning the monthly uptime percentage[14] to 99.9%. Any violation of the contract is penalized, hence service credit is transferred to consumers from the service provider.

According to [Sta14], there are basic steps of initializing and completing a SLA contract, which can be seen at Figure 2.2. Although the concept of SLA composition depends on both service provider and consumer, in cloud computing environment it is mostly dictated from service provider side. For example, consumer can not ask for 100% uptime of services. Therefore, the SLA templates are most predefined from the service provider side rather based on the negotiation from the consumers side.

[12] https://aws.amazon.com/ec2/sla/
[13] https://cloud.google.com/appengine/sla
[14] https://gsuite.google.com/terms/sla.html

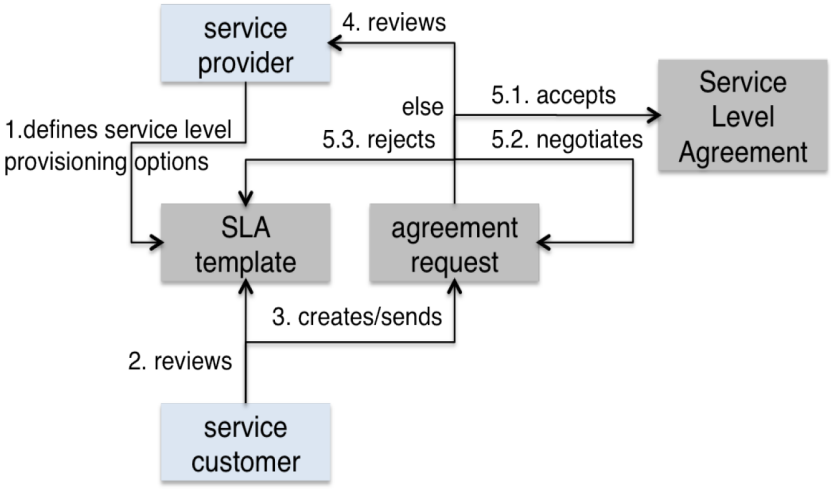


Figure 2.2: SLA initialization lifecycle [Sta14]

2.3 Cloud Application

Generally, cloud applications are hosted as SaaS fashion where stakeholders are: users, application owners and cloud infrastructure providers. Broadly, we can classify cloud applications in two categories. Firstly, the user facing applications which are highly delay sensitive and whose downtime or low performance can reduce productivity and revenue for application owner are regarded as *Interactive applications*. In contrast, the applications whose workloads are computationally bounded bag of tasks (BOT) and delay insensitive having a deadline, can be coined as *Batch applications*. Nonetheless, each category of application has several sub-category types and different metrics of interest. According to [IZM+16], there are basically 9 sub categories of applications which fall under above classification. Table 2.1 summarizes the metrics of interest for all the application types. Multi-tier web applications and business critical applications are the classical example of interactive applications where low-latency, reliability and availability are the most important and desired metrics to satisfy users need. For example, e-commerce shopping services like Amazon[15], e-bay[16] etc. are the popular multi-tier web facing application, whereas Monte Carlo simulations, financial and other types of modeling applications can be seen as business critical application. Due to the adoption of cloud technology, some applications such as online gaming, partial processing, and cyber-physical applications are emerging as new class of interactive application in cloud. These days, web applications and social networks generate enormous amount of data and there is a compelling need

[15] https://www.amazon.com/
[16] http://www.ebay.com/

to process such data in real-time. With the advent of cloud technologies like Apache Storm[17] and Apache Samza[18], several companies like twitter and LinkedIn using user data and web logs to provide near to optimum real-time analysis which enhances user experience and acts like an interactive, responsive and expressive application. On the contrary, compute intensive batch processing workload consumes lot of cpu resources *e.g.*, parallel implementations of scientific and engineering applications. Apart from that, there are plethora of applications using MapReduce paradigm of Map and Reduce tasks for big data processing, image processing and machine learning.

2.4 Energy management problems and opportunities

Data center is a warehouse facility composed of IT equipments (networked computers, storage etc) to host Cloud applications. With the adoption of cloud computing, the energy consumption of data centers have increased higher in magnitude. Since, it became a serious issue, researchers across the world have put much concern in last decades. These concerns can be classified in two categories: (1) How to reduce energy consumption to increase resource efficiency? (2) How to govern underlying power infrastructure for better energy management?

(1) How to reduce energy consumption to increase resource efficiency? Since energy consumption in data center is upsurging, using energy efficient hardware and software systems leave no choice. While virtualization technique enabled the adoption of cloud, it also facilitates to indulge energy efficient and reduction policies. Virtual machine (VM) migration [WJSVSY07] is one of the popular techniques to reduce number of used server via server consolidation, not only helps to reduce cost but also to reduce unnecessary energy usage. Also, when a server is under utilized, the energy effciency reduces significantly since it consumes more power, proportional to the task it is associated with. Therefore, by increasing server utilization also helps to increase resource efficiency [HAB07]. While switching off servers can conserve some energy it can have negative effects on servers overall energy consumption as well. Rebooting and turning off a server shows a large spike of power consumption. Therefore, research is also going on to find solution of when and what servers can be turned off and utill what time a turned off server should not be turned on to lower overall energy consumption [CASS15]. Since most modern servers have different power/sleep states, it is also an option by passing through these different states, rather than turning off servers completely. At low power state, a running server do not provide their full capacity, but changing from one power state to its next is much faster than switching turned off server to active mode [GHBAK12]. Some researchers applied dynamic voltage and frequency (DVFS) technique [HKBB09] which is an efficient

[17] http://storm.apache.org/
[18] http://samza.apache.org/

Table 2.1: Cloud Application types

Category	Application types	Metrics of Interest
Interactive	Multi-tier Web	availability,reliability,throughput, response time
	Business Critical	latency, throughput, system load, risk score
	Data Stream Processing	latency, throughput
	Online Gaming	response time, cost of operation, variability, reliability
	Vehicular Application communication	autonomy, shared info, system health
	Partial and delayed processing	response time, cost, reliability
Batch	Compute intensive Batch Processing	throughput, bounded slowdown/makespan
	Data intensive Batch Processing	jobs per minute, IOPS, cost, success/failure
	Workload generated by data center	availability, cost, checkpoint, recovery time

technology to control the processors power consumption, hence building scheduling algorithms [VLWYH09] based on DVFS allows to reduce further energy consumption by increasing resources efficiency.

(2) How to govern underlying power infrastructure for better energy management? Primarily, data center gets their power from the power grid. Apart from that, Diesel Generator unit (DG) is used as secondary backup and UPS units are used as intermediary upon a utility failure. In the wholesale electricity market, generally three types of electricity plans are offered through various provider [EML+12]: fixed pricing, time of use pricing and dynamic pricing. Like any large power consumer, data centers tend to adopt dynamic pricing. Recent data centers also employ energy storages to store energy when the electricity price is cheaper (because of dynamic pricing) or extra green energy if there is an on-site renewable plant integrated to data center. The integration of on-site renewable sources are becoming extremely popular, even though the initial cost is very high. However, any kind of renewable sources are intermittent, hence data center can not solely rely on them as a primary source of energy. Therefore, another option is to transport green energy from off-site renewable plants. Since the societal consciousness and sustainability practices are increasing, data center owner also looks for reducing carbon footprint by using these renewable sources by integration. Government and non-profit organization also put pressures the data center owner to achieve carbon neutrality or net-zero carbon emission. Carbon neutrality, refers to achieving net zero carbon emissions by balancing a measured amount of carbon released with an equivalent amount of renewable energy directly (on-site) or indirectly (offsite) usage or buying enough carbon credits to make up the difference. Carbon credits can be purchased through Renewable Energy Certificate (REC) and with the engagement of Power Purchase Agreement (PPA)[RWUS12]. The latter is a contract between a consumer and a renewable energy producer which allows the consumer to purchase a portion or all of electricity generated by the producer at a negotiated price for which it accumulates some form of credits such as REC. Big internet company like Google is offsetting their carbon footprint through REC and PPA[19]. Net-metering is also an option for data center, in which the excessive and unused green energy can be feeded back to grid to trade with brown energy. However, net-metering is only possible if the abundant green energy is stored in the energy storage, hence losses incures due to the voltage transformation involved in feeding the energy back to the grid [GLN+12]. Figure 2.3 shows different direct and indirect energy integration options in a data center. Additionally, countries like UK, Australia and USA have already developed and regulated carbon capping policy by introducing cap-and-trade and carbon-tax [GS13]. A cap-and-trade system sets a maximum level of pollution, a cap, and distributes emission permits among companies that produce emissions. Any large power consumer or data center can obtain this permits either through an initial allocation or auction or through trading with other company. On the other hand,

[19] https://www.google.com/about/datacenters/renewable/

a carbon tax scheme imposes tax on each unit of greenhouse gas emmissions and provides incentives if the emmissions is less therefore, lesser taxes. Although carbon tax scheme is yet to be regulated at France [20].

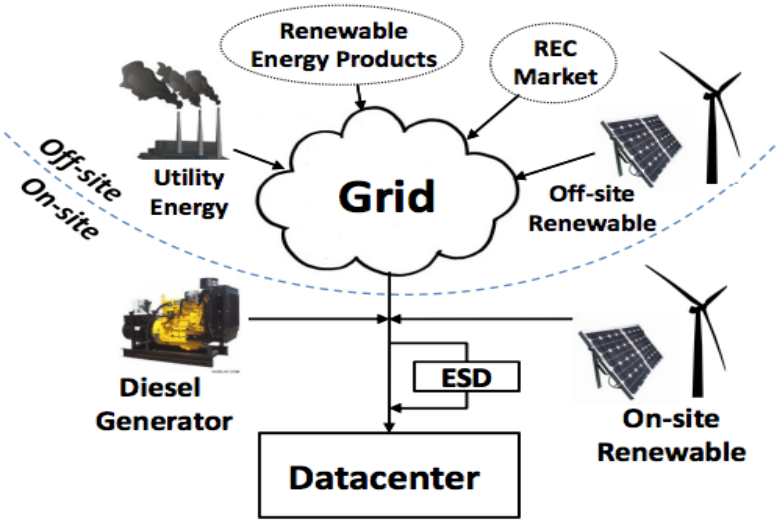


Figure 2.3: Various energy integration option in data center [RWUS12]

2.5 Summary

This chapter discusses SLA in different layers of Cloud computing and the heterogeneity in the Cloud applications. While metrics of interest in Cloud application depends on their characteristics, one SLA fits all can not be enough. On the other hand, it is difficult for Cloud providers to propose differentiated SLA for specific application. Therefore, looking for the interest and common ground for both parties (Cloud providers and consumers) can be a pragmatic solution while offering new class of Service Level Objective (SLO). Later, we highlight the technologies and techniques behind the reduction of energy consumption and energy management poilicy for further reduction and bettering the eco-system. All the notions used in this chapter will ease the reader to better understand next chapters.

[20] http://www.reuters.com/article/us-france-carbon-idUSKCN12K2OG

Chapter 3

Related work

The objective of this thesis is to smartly manage renewable energy in Cloud computing environment from infrastructure to application to reduce carbon footprint. Firstly, this chapter provides some literature review on greening the cloud computing infrastructure through SLA specification and efficient energy management. This leads to discussion on identifying the new possibilities and techniques to be considered for proper energy management at data center. Later, we sketch out some relevant and selected work on how different kind of Cloud application can be energy aware but still can perform at a targeted level so as to adapt with green energy profile. Furthermore, we discuss the opportunities and challenges to adapt Interactive Cloud application based on different level of adaptation techniques.

3.1 Greening the Cloud computing backend environment

The back end platform of Cloud computing is comprised of servers, storages, cloud based delivery and network models, which are hosted on data center. Cloud service providers are charged with ensuring the performance, availability, hence they require mechanisms to agree on quality of service provision which is usually established through SLA. In today's terms, energy dependent SLA can emerge as a key concept to ensure the energy efficiency of the data center as an on going activity, rather as a one-time event. The first subsection 3.1.1 highlights the recent works related to energy aware SLA.

What's more, todays large-scale data centers consume tremendous amount of power and impact high carbon emission. Therefore, to limit the carbon footprint, greening the data center leaves no choice. Over the past decades, academicians and industrial research ventured more on bettering energy-proportional computing technologies, efficient power delivery and cooling systems to reduce energy consumption and associated costs. However, energy efficiency alone will slow down the process of reducing carbon footprint in data center. Therefore, recent efforts have concentrated on powering the data center

with renewable sources and exploiting the greenness nature of these sources to reduce further carbon footprint. To be concise, a green data center should have two elements: energy efficient capabilities and presence of green energy [DLJ+14]. On the other hand, data centers humongous power cost is the primary concern for cloud service providers. Hence, dynamic and economical energy management in data center becomes a key concern which can eliminate energy incurred costs and carbon footprint by lowering the power consumption and/or utilizing green energy. The second subsection 3.1.2 provides recent efforts made in this area.

3.1.1 Green cloud through SLA specification

Laszewski et al. [vLW10] investigated different green IT metrics that considered environmental impacts as part of the SLA for building GreenIT-as-a-Service (GaaS) which can be reused as part of a SaaS and IaaS framework. The framework provides how green metrics (i.e., DCiE, PUE, DCeP, SWaP) can be taken into consideration to reduce energy consumption in different layer of Cloud computing to provide greener services. The metrics can help service providers decreasing environmental impact without the knowledge of users. However, it is important to explicitly develop contracts by specifying these metrics.

Klingert et al. [KSB11] [BKT12], first introduced and defined the notion of *GreenSLA*, focusing more on optimizing energy per job based on known hardware and software techniques. While performance aware SLAs have higher priority and importance for service providers, GreenSLA can provide new degrees of freedom to re-organize their service provision in an energy efficient way in a data center. Thus, the authors propose GreenSLA alongside regular SLAs, using eco-efficiency as a differentiating factor. Figure 3.1 shows the difference between traditional SLA and GreenSLA, where the latter tries to optimize all elements of the triangle objectives. Furthermore, their research work is part of *FIT4Green*[1], which aims at combining several energy saving software and infrastructure strategies on top of data centre management system. For software optimization, a Job Subscription Definition Language (JSDL) is used to explain the scheduling of batch jobs based on GreenSLA. For example, *Energy EfficiencyClass A* accepts higher error tolerance (.05% instead of .01%) and longer makespan of running jobs compared to *Energy EfficiencyClass C*. Therefore, customers can have 10% price rebate compared to regular SLA and further percentage of reduction can be offered if they accept delaying of the deadline. For data center level optimization, if service providers have several data centres, authors proposed to schedule jobs in priority basis to lower PUE valued data center.

Later Atkinson et al. [ASK14] provided green specification by including services exact functionality in GreenSLA. The specification is consist of three view of service, namely: structural, functional and behavioral views based on Graphical UML models. Structural view defines service properties including each data classes annotated with eco-constraint with stereotype «env». Whereas, Functional view enhances the specification with QoS

[1] A project funded by European Union. http://www.fit4green.eu/

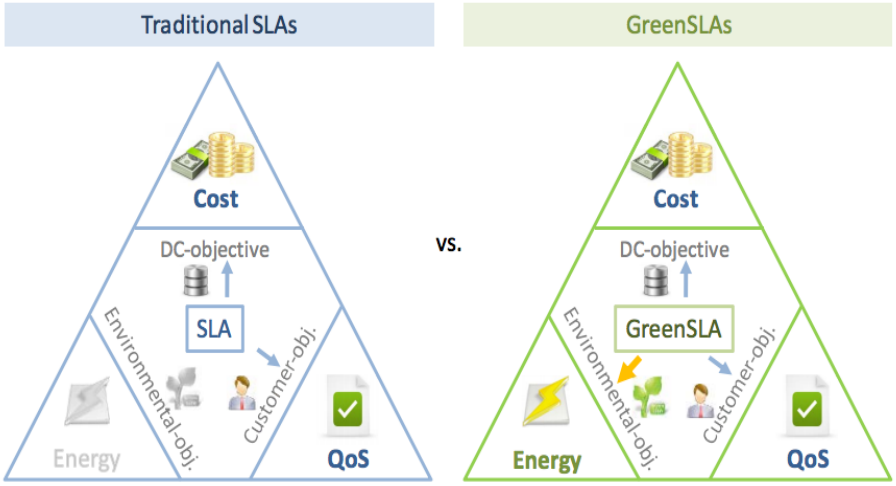


Figure 3.1: Traditional SLA vs GreenSLA [BKT12]

constraints. For example of a online DVD store, it specifies that the execution of play must take no more than 0.005 percent of the available bandwidth with cost and availability constraint. On the other hand, behavioral view adds the information about probabilities of the various state transitions taking place. While the novelty of the GreenSLA is well understood, the research work ignores how to capitalize green energy information in the SLA specification.

To this point, Haque et al. [HLG+13] considered an SLA that specifies the proportion of green energy that the IaaS provider should use to run the job (e.g., x% of the job should run on green energy). Therefore, they introduced a new class of SLA along with traditional SLA based on green energy. Each client can indicate the percentage of green energy they need to run their job, which authors have described as Green SLA. On the other hand, provider can earn extra monetory unit (*e.g.*, premium) for meeting Green SLA but penalized if they violate the contract. To justify the SLA, they created two separate server cluster with two different power buses, one with green energy and another with mix of energy. Based on the green energy requirement per job, the proposal formulate a scheduling plan by constraining which job should spend how much time in the green cluster. However, their proposed Green SLA approach is per application/job specific, where every job can have different green energy requirement. Additionally, providers reject job admission if there is lack of green energy. Nonetheless, giving the freedom to user to specify random green energy requirement can complicates the SLA specification and enormous amount of template generation is required to negotiate between consumers and providers. Moreover, lack of formalized SLA affects the set up of the enforcement process to be automated and validated, hence hindrance consumers with burden.

Wang et al. [WLDW10] proposed green SLA based on energy aware scheduling of resources by exploiting dynamic voltage and frequency scaling (DVFS) technique. The idea is to extend the job execution time for non-critical jobs and reduce the energy consumption without increasing the task's execution time as a whole. Their green SLA contract definition phase creates some green SLA templates where user might accept some performance or QoS degradation in order to reduce power consumption for their task completion. The templates are based on response time, CO_2 emission and power consumption. The results shows that, if service provider can negotiate with user to have larger job execution time, larger amount of energy can be saved. Therefore, these specifications are ought to build green services.

Likewise, Ahmed et al. [AFZZ+14] [ALFZ+15] addressed Green SLAs where service providers (application owner) specifies carbon emission constraints along with traditional resource requirements to cloud infrastructure provider in Virtual Data Centers (VDCs), i.e., a set of VMs and virtual links with guaranteed bandwidth. Authors proposed two metrics: (1) carbon emission per unit of bandwidth (tonCO2/Mbps) and (2) carbon emission per core (tonCO2/Core). To enforce Green SLA, cloud provider monitors and computes the carbon footprint of each VDC request. After each monitoring period, SLA is enforced. In case of violation of specification (i.e., the carbon footprint for the VDC is higher than the limit specified in the SLA), cloud provider pays penalty to service providers. Again the work ignores the formal SLA specification and enforcement model and lacks real use case in any business model.

3.1.2 Greening data center through energy management

As different kind of renewable sources are intermittent in their own way, it is important to formulate a capacity planning to maximize their usage in a data center. Apart from that, the price of electricity varies from region to region that gives opportunity to the data centers owner to exploit the price depending on location diversity. To this, [RLXL10] proposed to exploit the price diversity in Internet Data centers to minimize electricity cost. Afterwards they formulate the minimization problem having delay guarantee for workload as a constraint. Similar to the approach, [FYH+15] proposed a stochastic optimization based approach to reduce power cost for delay tolerant workloads. Their approach showed, with the increase of service delays, power cost could be reduced largely.

Later, authors at [BR11], proposed *ReRack*, a power simulation framework for data centers considering multi-source energy environment. The idea is to reduce the cost related to power by carefully analyzing the power demand and power supply from different sources. In summary, the simulator follows like: *"Calculate power supply from renewable sources and calculate demand, if power is surplus, store it to batteries, if not, draw the remaining power from the grid, measure how much workload was satisfied and determine how to power up machines for next period"*. Traditionally, wholesale electricity market can have different energy prices at same location. For example, in California, electricty price fluctuates in

every 15 minutes. To take the opportunity of the price fluctuation, authors at [UUJNS11] proposed cost reduction in data center by using uninterrupted power supply (UPS). The basic idea is to charge the storage when the energy price is cheaper and opportunistically recharge them to the computing servers during high price period or for peak shaving.

Unlike the above contribution, authors at [DLJW13], [DLJ+13] investigated two time scale energy market (*e.g.,* long term and real time) to reduce energy cost. The proposal is to buy energy in advance from long term ahead market where prices are cheaper and if the demand of energy increases in real time, consume the energy from the grid or from the on-site renewable sources. Usually energy prices are higher in real time market. The authors considered that the delay tolerant workload can be moved to next time slot if the current energy price is higher or stored energy in the ESD is not sufficient. To do that, they provide an optimal online algorithm called *SmartDPSS*, which activates in each 15 minutes to compute the difference between supply and demand and to determine cost-effective solution of selecting appropriate energy sources while minimizing the cost in the long term. Their solution claim to not require any priori knowledge in system dynamics. However, without the demand forecast, it is not possible to do energy procurement from the long term market.

While the above works consider efficiently managing grid-tied brown and on-site renewable energy sources to do cost reduction, other implicit options exist in energy market. Authors at [RWUS12], discussed on how explicit and implicit integration option in a data center can affect on the cost reduction. Implicit integration consists of (1) off-site renewable energy which should be transported across the grid to data center; (2) buying REC certificates or investing nearby renewable energy plant. Given all these choices, authors formulate an optimization plan to reduce carbon footprint while lowering the energy cost. Interestingly they provide some key insights. Firstly, by smartly using on-site renewable energy for peak-shaving can reduce cost significantly and also can replace or reduce the need of ESDs. Therefore, by reducing the peak shaving cost, the OPEX cost for on-site plant can be recovered. One research analysis [GBL+11] indicates that, by peak-shaving via on-site renewable source can recover plant cost in 8-9 years. Secondly, on-site renewable sources becomes less cost-effective if the plants capacity factor is below 24-25%. Thirdly, to reduce large amount of carbon footprint (*e.g.,* beyond 30%), off-site renewable energy usage becomes more cost-effective. Therefore, hybrid use of the two options is the most cost-effective across the spectrum.

Apart from this work, authors at [LWL+14] proposed *GreenWorks*, a power management framework for green high-performance computing datacenters powered by renewable energy mix. GreenWorks provides a framework for managing datacenter power across several layers from datacenter server to onsite renewable energy mix. It comprises of two elements; green workers and green manager. Green workers are power optimization modules that uses supply/load control strategies for different types of renewable energy systems, wherease green manager manages the hierarchical coordination between these workers. They propose three types of green workers; (1) base load laborer responsible for

tracking the difference between power supply and demand, (2) energy keeper regulates the use of stored energy in an efficient manner to satisfy workload performance and (3) load broker opportunistically increases the servers processing speed to consume surplus energy from the renewable sources. Their result shows that, by efficiently using energy storages, battery lifetime can be extended to 23-24%.

Recently, authors at [KL16] introduced a framework that optimally selects energy sources and determines their capacity, which aptly balances energy sources, grid power and energy storage devices in terms of cost, emission, and service availability. Their envisioned energy components in data center is shown at Figure 3.2. Compared to all the work we discussed before, this framework considers net metering, cap-and-trade and carbon tax (popular economic incentives to control carbon emission) along with traditional constraints (*e.g., cost reduction, service delay etc.*) in their formulation. Their key insights suggests that, energy storage give very little help on cutting lifetime total cost or emission. Additionally, they realized that, using multiple energy sources can significantly reduce data centers lifetime cost and dependency to the grid. Additionally, power cost can be further reduced if the net metering program increase the capping limit.

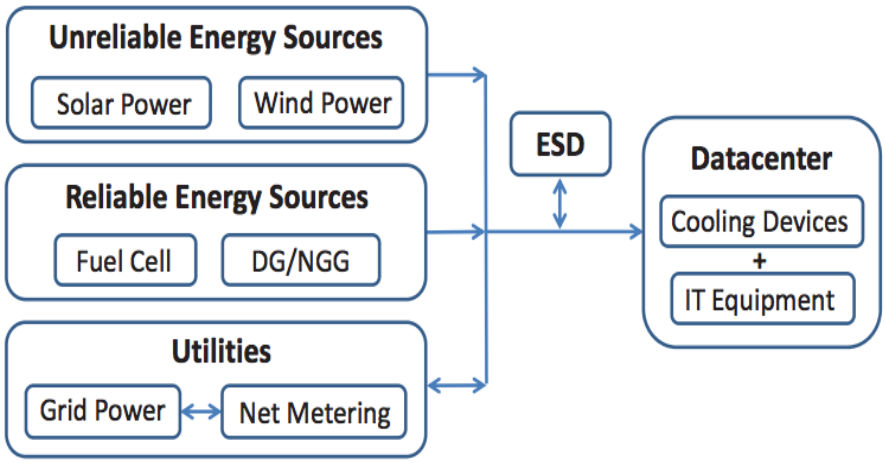


Figure 3.2: Data center powered by multiple energy component [KL16]

3.2 Discussion

The section 3.1 presented a selection of relevant work about greening the cloud environment and data center in terms of SLA specification and energy management. We refrain ourselves from discussing and sketching out energy efficient enabling technologies (*e.g.,* DVFS and various sleep states for servers) and techniques (*e.g.,* VM migration, server consolidation),

since these methodologies are widely adopted, matured and practiced for several years now. Our goal is to look further ahead to reduce carbon footprint by efficiently managing different energy sources that can be imposed from user side to the grid authority. In order to evaluate the works described earlier, we summarize the research efforts by defining some qualitative comparison attributes:

- **GreenSLA:** As demand for green products are ever increasing, users are more conscious about the greenness of the product, be it in the supermarket or in software and hardware system. Therefore, specification of SLA is required. Usually in the literature, Green SLA notion is used to relax some performance requirements to reduce energy consumption.

- **Energy management:** Generally, cloud providers does energy management to reduce the energy cost that accounts to approximately 20% of the total cost [GHMP08]. By reducing costs, the target is to mitigate emissions in the operating phase. However, choosing the price and location diversity (proposed at [RLXL10], [FYH+15]) can not guarantee lower carbon emission because, grid's emission factor may be high at that time or at that place. Therefore, we categorize the energy management based on the presence of brown energy and mix energy (brown and green energy).

Table 3.1 summarizes the Section 3.1 with regards to aforementioned attributes. To propose GreenSLA, some authors in [vLW10], [KSB11], [BKT12], [ASK14] provides the SLA specification and requirements needed to establish Green SLA. The idea is to increase the job execution time or to schedule the jobs outside of the peak power time. On the other hand, few others [WLDW10], [AFZZ+14], [ALFZ+15] used the notion to increase the energy efficiency inside the data center which is not explicitely specified to the user. The only work that relates GreenSLA to the presence of green energy is [HLG+13]. However, the authors ignored the SLA specification phase, associated validation and penalty model that is necessary in any service based platform. On the other hand, from the Table 3.1 we can see that most of the works, [DLJ+13], [DLJW13], [LWL+14], [KL16] proposed multi-source energy manangement without taking into account the renewable energy prices and ignores the impacts of using energy storages in their problem formulation. Thus it hampers the infrastructure providers to envision a green data center where they can offer green computing resources to application owner having a formal contract with partially or fully powered by underlying clean energy sources.

After rigorously analyzing two very different doamins of work, we realize that energy management and SLA can be linked together. Therefore, in this thesis, Chapter 4 and 5 investigates the opportunity to exploit the energy market while propose and revise the notion of GreenSLA. Our idea is to propose new class of explicit SLO mentioning the percentage of green energy provided along side with computing service by managing the underlying energy infrastructure and multi-source energy market. By doing so, users

and application owners can have the possibility to host their application in an explicitly expressed green cloud environment having formal contracts.

3.3 Energy and Performance aware cloud application

The increasing enthusiasm and consciousness of reducing energy consumption leads to smarter ways to consume energy in cloud data centers. While an efficient energy management technique in data center can reduce unnecessary use of brown energy and better utilize green energy without going to waste, smarter ways of consuming energy by an application can further reduce carbon footprint. This section provides insight on how two major category of applications (*i.e.*, batch and interactive jobs) can opportunistically and smartly adapts themselves based on green energy and performance requirements.

3.3.1 Opportunistic scheduling of Batch jobs

As the interests and trends have been growing to implicit or explicit integration of green energy to the data center, many industry and academia people putting more efforts in developing green energy aware algorithms and systems. Energy consumption reduction can impact on joule efficieny [NHL16], that is, the work done using per joule. On the other hand, energy efficiency may hinder the opportunity to consume green energy when it is available [SS09]. Therefore, energy efficiency as well as green energy awareness are both equally important. To this, we highlight some recent work in this area in the following paragraph.

Green Energy-aware approaches. The authors proposed *GreenSlot* [GBL+11], a parallel batch job scheduler for a datacenter powered by an on-site solar panel and the electrical grid. Based on the historical data and weather forecast, GreenSlot predicts the amount of solar energy that will likely be available in the future. Subject to its predictions and the information provided by users, it schedules the workload by the order of least slack time first (LSTF), to maximize the green energy consumption while meeting the job's deadlines by creating resource reservations into the future. If the job can not be scheduled in green energy availability period, GreenSlot schedules jobs for times when brown electricity is cheap. Therefore, the proposal seeks to maximize the usage of green energy in the data center while cost of brown energy usage can be minimized largely.

Later their work evolved into data processing framework by proposing *GreenHadoop* [GLN+12]. The idea relies on deffering background computations *e.g.*, data and log analysis, long simulations etc. that can be delayed by a bounded amount of time in a data center to take advantage of green energy availability while minimizing brown electricity cost. For instance, many jobs (*e.g.*, data and log analysis, long simulations etc.) in data center have loose performance requirements. On the other hand, datacenters are often underutilized due to low activity. As example, Hadoop keeps all servers active even if they are idle,

Table 3.1: Energy management and GreenSLA

Approaches	Energy management		GreenSLA (No Green Energy)		GreenSLA (Green Energy)
	Brown energy	Green energy	Specification	Implicit Notion	
[vLW10], [KSB11],[BKT12], [ASK14]	✗	✗	✓	✗	✗
[AFZZ+14], [ALFZ+15],[WLDW10]	✗	✗	✗	✓	✗
[HLG+13]	✗	✗	✗	✓	✓
[RLXL10], [FYH+15], [UUJNS11]	✓	✗	✗	✗	✗
[BR11], [DLJ+13],[DLJW13], [RWUS12], [LWL+14], [KL16]	✓	✓	✗	✗	✗

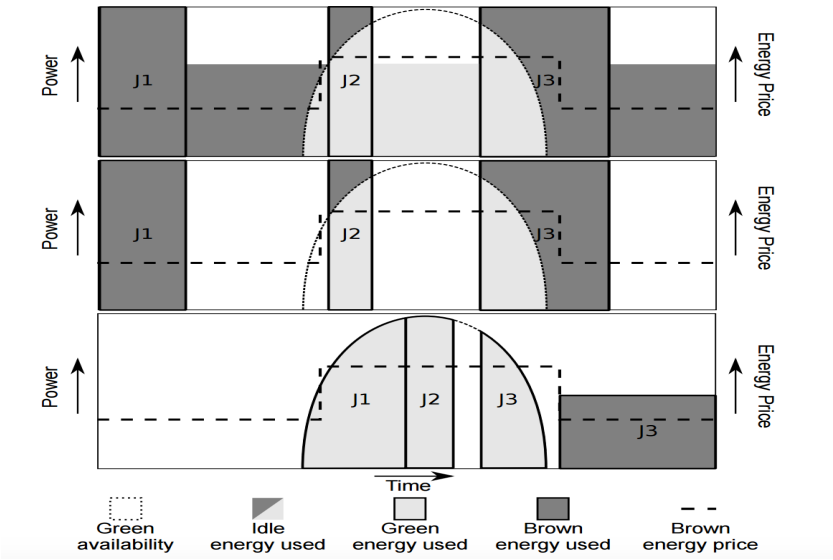


Figure 3.3: Scheduling 3 jobs using GreenHadoop [GLN+12]

which is illustrated at the top of Figure 3.3 for three MapReduce jobs. Following are the behavior of an energy-aware version of Hadoop (middle) and GreenHadoop (bottom). In contrast to conventional Hadoop, GreenHadoop used as many servers to match with green energy supply and delayed jobs J1 and J2 to maximize the green energy consumption. As the Figure 3.3 shows, it executed part of J3 with green energy, and delayed the other part until the brown energy became cheaper. Therefore the idea is to run fewer servers when brown energy is cheap, and even fewer (if at all necessary) when brown energy is expensive. In conclusion, their proposal leads to operating few hadoop clusters when green energy is scarce.

Similar to this work, authors [HGRN15] proposed *GreenPar*, a scheduler for parallel high-performance applications to maximize using green energy in a partially powered data center and reduce brown energy consumption, while respecting performance aware SLA. When green energy is available, GreenPar increases the resource allocations to active jobs to reduce runtimes by speeding up the processes while slow down the jobs to a maximum runtime slowdown percentage that is defined in SLA during the scarcity of the green energy. Interestingly, GreenPar avoids using energy storage by increasing resource utilization during the high availabity period of green energy. To conclude, *GreenSlot, GrennHadoop and GreenPar* can be an important software component in green data centers that run HPC and defferable workloads, to improve the sustainability in cloud computing paradigm.

The work [LCB+12] proposed a holistic approach that integrated management of IT workload *e.g.,* batch and interactive workloads, cooling *e.g.,* chiller and outside air cooling, and power sub-systems *e.g.,* renewable supply and dynamic pricing, to provide energy-

efficiency accomplishment in a data center. The solution relies on rigorous analytical modeling of each components mentioned earlier to formalize a generic yet useful solution. Renewable energy production, workload and electricty price is predicted in advance to create capacity planning (see Figure 3.4) at midnight for next 24 hours where total IT demand ratio between the interactive workload and batch jobs is considered as 1:1.5. Interestingly, interactive workload is simplified by considering only normalized CPU utilization from a trace without going to any intensive detailing.

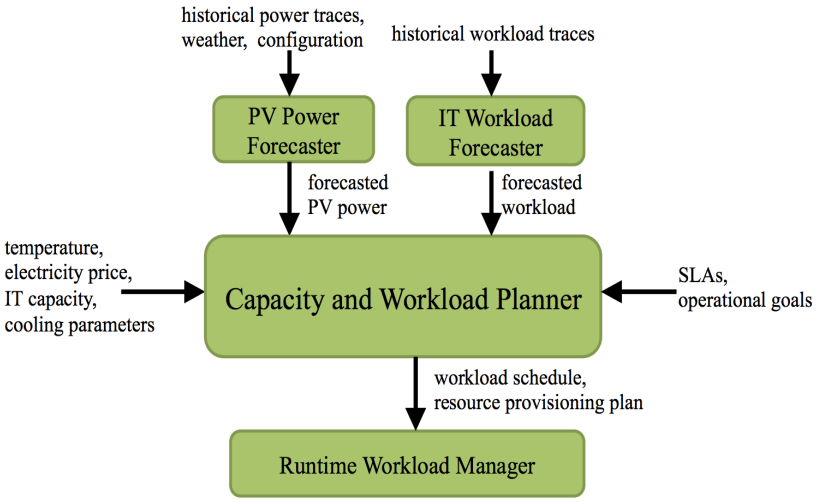


Figure 3.4: System architecture of capacity planning [LCB+12]

The authors [DSFH15] presented an Energy Adaptive Software Controller (*EASC*) to make task and service oriented application adaptive to renewable energy availability. The work was part of by DC4Cities project [2], which aimed at gathering renewable energy related information from energy providers and energy constraint directives from Energy monitoring authority (in context of Smart city) through an interface. Following the information, the PaaS layer is responsible to adapt the application by satisfying energy related constraints to consume more green energy, therefore building more eco-efficient policies for data center. Figure 3.5, better illustrates the objective.

Therefore, the authors proposed to forward the energy related information to PaaS level via an API, so that an optimization plan can be invoked which involves desired working modes of an application considering energy and SLA constraints. For cloud application adaptation, the authors have provided two different algorithms for task and service oriented applications respectively. For task oriented application, aggressiveness and eagerness factor are proposed in the algorithm, while former factor controls the possibility

[2] An European project on environmentally sustainable data centers for samart cities. Ended on 2016. http://www.dc4cities.eu

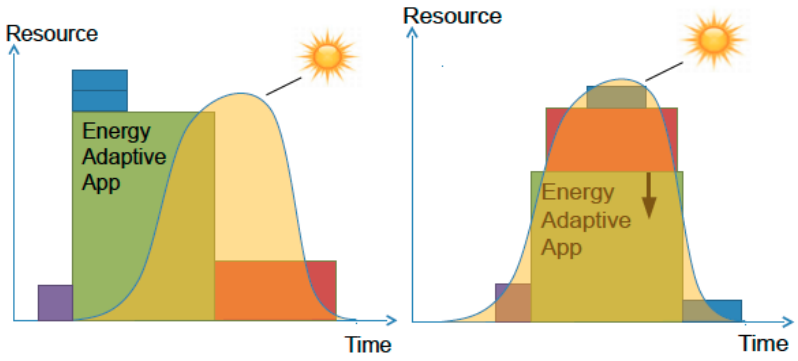


Figure 3.5: Adapting applications for a better usage of renewable energies [DSFH15]

of consuming more or less green energy, latter controls how much early or late the task could be finished. On the other hand, for service oriented application, their proposed algorithm first predicted the time slots corresponding to high green energy availability and then scheduled the higher working modes of the application allowing better performance than SLA. During the low availability of green energy period, lower working modes are applied which is below targeted SLA. However, service oriented application is defined as running web, database and mail servers and higher mode depicts multiple data center site is active with full capacity while lowest mode indicates running a single site with minimum capacity. For the experimental purpose, authors proposed 6 different working modes.

Unlike for big data centers, in [LOM15], the authors proposed a opportunistic broker framework named *PIKA*, to save energy in small mono-site data centers. Similar to other articles mentioned above, this work also considers scheduling batch jobs in the period of green energy availability. However, the notion of interactive job was provided by indicating that fixed number of web servers run all the time to process web requests. Apart from that, the work proposed resource overcommitment approach to increase ON servers utilization so that some servers can be turned off where VM's are under loaded. The authors introduced slack period to distinguish between web job (slack < 1slot) and deferrable batch jobs (slack > 1slot) to prioritize the web jobs. However batch jobs can not turn ON a server until unless their slack is strictly inferior to 1. At that point, any batch job is treated as web job, which is illustrated at Figure 3.6. Additionally, potential ON servers can be turned on when green energy availability is more than the current energy consumption. In conclusion, analysis of results showed that, PIKA can increase green energy consumption largely in contrast to any baseline approach, although over all consumption can be more than the baseline.

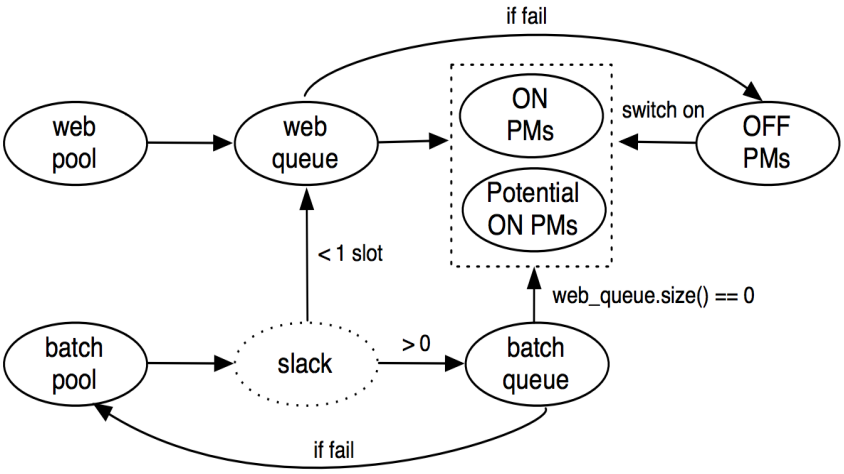


Figure 3.6: Placement for batch and web job [LOM15]

3.3.2 Cost-aware approaches in geo-distributed cloud

Due to the intermittent nature of renewable energy sources *i.e.*, wind and sun, they tend to be more expensive than brown energy. For example, the industrial electricity price for solar energy can be 16.14 cents per KWh in a sunny climate and 35.51 cents per KWh in a cloudy climate [3]. Unfortunately, Most of the research articles are unware of the cost analysis due to this hidden and variant nature of prices. Hence, most work considers green energy incurs $0 [WZL14][APKSG14]. Authors in [TB15], described a fuzzy logic-based load balancing method to dispatch user requests to most suitable data center where least cost incurs. While most of the research considers workload and green energy prediction [APKSG13] to modelize the problem, the authors used fuzzy logic which requires no priori future knowledge. Similar to previous works, they also ignored the renewable energy cost in their problem formulation. Apart from that, some works [APKSG14], [GDXHJ14], [GZG+16] explored the dynamic pricing of energy market in geo-distributed cloud to reduce electricity costs incurred by service provider.

Nevertheless, utilizing renewable energy in data center can impose huge pressure to data centers owner due to the stringent budget constraints. For instance, energy bill incurs around 20% of total data center operations bill [GHMP08]. Therefore, how far efficiently consume green energy that does not incur higher than traditional cost, is an important research issue. To this, in [ZWW11], the authors proposed *GreenWare*, a middleware system that maximize the usage of green energy in geo-distributed cloud scale data centers by dynamically dispatching workload requests by following renewable, subject to energy budget constraint. The middleware performs three steps: computes the hourly energy

[3] Solar Electricity Prices. http://solarbuzz.com/

budget and historical behavior of workload, runs an optimization algorithm based on constrained optimization technique, lastly dispatch requests according to optimization plan, which is illustrated at Figure 3.7. The workload was modeled using queuing theory, which only captures arrival rate, waiting and service time of a request. Moreover, traditional cloud application possess higher complexity in practice. While comparing their approach with a greedy green approach, which tries to consume as green energy as possible, GreenWare can decrease around 29% electricty bill while green energy usage can be at acceptable level. Therefore, they address the criticial issue by justifying a trade-off between maximizing green energy usage and energy budget. Similar to this, [GCWK12] proposed a flow optimization based framework for request-routing considering the trade-off between access latency, carbon footprint and electricity costs to upgrade the plan of choosing data center in specific intervals. Results show that, 10% carbon emission can be reduced without increasing mean latency or electricity costs.

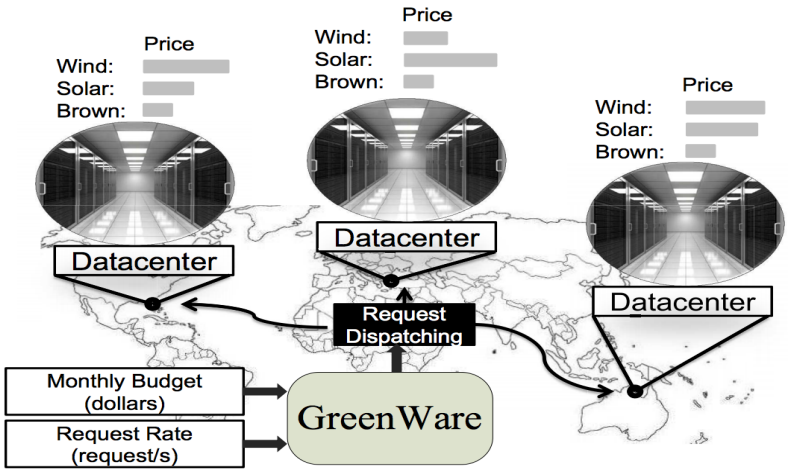


Figure 3.7: Middleware for distributed cloud-scale data center [ZWW11]

Similar to this, authors at [CJX14], [LLW+11] provided capacity planning method for geo-distributed cloud to minimize system energy cost. Unlike the previous works, authors at [CHT12] proposed a scheduling algorithm called *MinBrown* which considered the cooling dynamics. Usually, when the solar energy is available, the outside temperature is higher and cooling requires more effort to cool the data center. By adding this new constraint, the algorithm made schedule plans for batch jobs with slacks to choose best data center in each time frame based on constrained optimization. Results showed that, the ratio of brown energy can be reduced compared to green energy. In conclusion, all the works mentioned above tries to consume green energy without incurring additional costs or satisfying predefined budget constraints.

3.3.3 Self-adaptiveness for Interactive Cloud application

Performance and availability are the foremost criteria for web facing interactive cloud application. Nonetheless, poor performance (*e.g.*, low throughput and high response time) can lead to service saturation, which might results disruption of availability of service. On the other hand, by maximizing availability with high performance can incur higher revenue for service provider. Since, primarily performance metrics dictates the availability and higher revenue term, we discuss some of the relevant and well known research work in following sub-section.

3.3.3.1 Performance aware approach

Performance of an application can be guaranteed in several ways which also involves several layers of cloud computing paradigm. To provide better performance, functional and non-functional adaptation is required. Functional adaptation can be seen as the capability of an application to self-adapt by re-organizing their architectural change on run-time environment or by activating/deactivating loosely couple components to the changing condition. Whereas, non-functional adaptation is more concerned about efficient online resource management plans and techniques mostly dealing with infrastructure to prevent transient behavior of hosted applications to changing conditions *i.e.*, workload surge, hardware failure, intereference between colocated application [MTS+12] etc.

Functional adaptation The authors at [AB99], introduced the notion of degradation in online services for the very first time to improve server overload performance. Server overload may occur either due to saturation of CPU, bandwidth or due to saturation of the communication link capacity connecting the server to the network. Their contribution lies on preventing both types of overload via content adaptation. During the overload condition for an e-shopping site, the approach switches to provide JPG image instead of GIF image to prevent service outage, where latter image is of high quality with larger sizes in kilobytes. Later, same authors [ASB02] proposed a promising analytic foundation based on classical feedback control theory to achieve overload protection and performance guarantees by service differentiation. Keeping CPU utilization of the server under a certain threshold *i.e.*, 58% (can be varied depending on application) was the primary objective to keep the response time of a web application under a desired limit. While the utilization is higher than the set point, the quality of content is degraded to prevent high service delays to the customers. In summary, the degradation was realized by data filtering and compression of images in online services by following resource utilization. However, to achieve degraded level of services, the contents must be pre-processed a priori and stored in multiple copies that differ in quality and size.

Later Philippe et al.[PDPBG10] proposed component based representation of resources for building distributed systems capable of autonomously deciding when and how to

adapt the service level. Their solution can find the most resource-intensive component/-components in the system at any given time and characterizes these list of components based on their path in the system's structure. Therefore, it allows to identify promising adaptations by targeting the hot paths. In case of overload, to keep the service level intact, their approach consists in choosing most inexpensive computational components *e.g.,* based on cpu intensiveness in the architectural path in priority to the components that actually cause the overload. Furthermore, this inquisitive nature of architectural self-adaptation was applied in a component based e-library application which is shown at Figure 3.8 to validate the authencity and reliability of their approach. To further validate their approach, authors used a real-world web application *i.e.,* RUBiS with emulated workload to find computationally expensive architectural path. Based on the performance profile, their proposal was to disable sorting options in the SearchByCategory and SearchByRegion components, which is usually a cpu-intensive operation. Similar to [AB99] and [ASB02], these approaches require prior knowledge of the platform, offline performance modeling and building cloud applications from scratch.

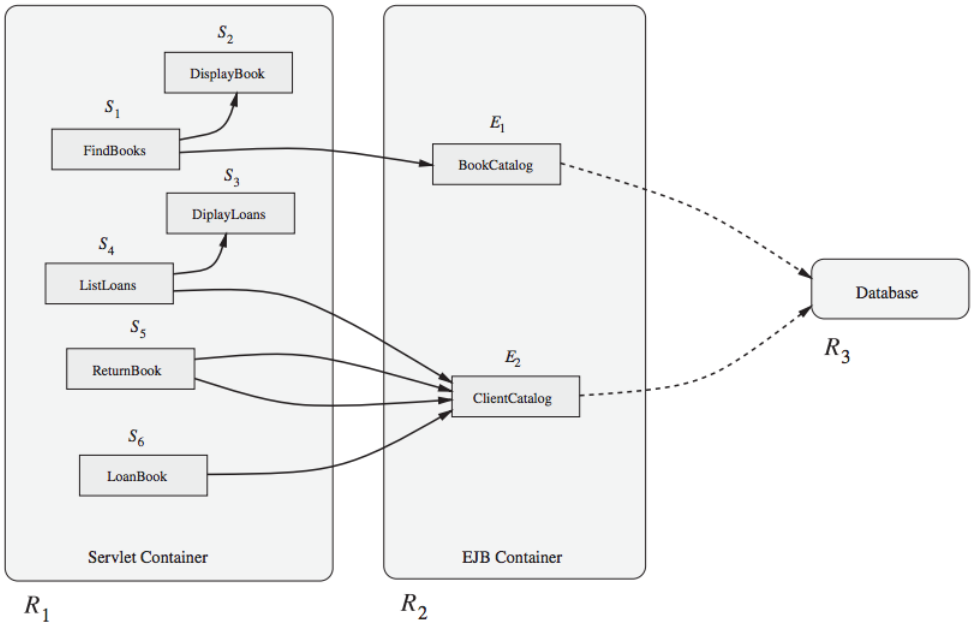


Figure 3.8: Component based representation of an Application [PDPBG10]

Likewise, authors in [CFF+11] proposed E-BP, an energy and QoS aware business process which is an extension of regular BP model considering different strategies consisting of execution model of business tasks. Traditionally BP applications are hosted in a virtualized infrastructure and can be monitored and measured at fine-grained way, be it response time, task completion, energy consumption etc. Based on the measurements, authors provided

Green Performance Indicators (GPIs) and QoS metrics with possible set of constraints, that has to be satisfied by functional requirements. Depending on the metrics, a set of adaptation strategies are proposed to maximize the objective function. For example, less functionality strategy refers to a computational or data degradation *e.g.,* skip a task or task functionality; less quality corresponds to QoS degradation by turning off a service of a task if there is more than one service available for same task. Moreover, aforementioned adaptation can be enacted along with the other adaptations at multiple layers (application, middleware and infrastructure). However, if the number of application increases, the number of dependencies and constraints among different metrics can increase proportionally, which may lead to over-constrained rule definitions.

Harmony [CIAP12] framework was proposed to reduce probability of stale reads in cloud storage by tuning the consistency level at runtime in a self-adaptive manner. Popular cloud storage systems allow modern web facing real-time applications to scale up their systems for maintaining performance with very high availability. Scaling up the data storages can leads to data inconsistency across the replicated instances which can limit availability and increase the latency to retrieve the data. In contrast, providing strong consistency all the time by means of synchronous replication can result bad performance. To this, the proposal gradually and dynamically scales the consistency level to best suit the application requirements, while taking into account the system state. For a running workload, the authors tried to estimate stale reads rate in a storage system by probabilistic computations. This approach is the opposite of strong consistency in data storage system *e.g., cassandra*, where read operation must wait for all the replicas to reply with consistent data in order to return the data to the client. Therefore, the motivation to introduce the adaptation mechanism is to reduce the cost for strong consistency model which can lead to unavailability and high latency to the user side. Towards this goal, authors find adequate tradeoffs between consistency and both performance and availability while cost reduction is eminent.

Klein et al. [KMAHR14] first proposed Brownout in software application to allow applications to avoid saturation in a robust manner in case of unexpected events occur. They borrowed the *Brownout* term from electrical grids, where voltage can be dropped intentionally to prevent blackouts through load reduction. Web facing application provider and their underlying infrastructure provider can face several challenges *i.e.,* hardware failure, budget constraint, unplanned workload surge, etc. Given the scenario, overloaded application may trigger performance degradation and lead the applications to saturate, in which some clients can face high latency to no services at all. To solve the challenge, the authors examine adaptive adjustment of controller parameters which decide upon the execution of optional application parts dynamically in varying load and resource contention situations. The primary target of their solution is to keep response time to certain or given threshold while minimizing the tracking error to the setpoint. Unlike [PSZ+07], [KCH09], their model does not require offline performance modeling to build an input-output relationship. Rather, the impact of input is measured in each control

period to build a bare estimation on how the system is performing and how much additive correction is required to keep the system in a targeted state. Their model also describes pole placement which determines how fast or slow the controller reacts to the error in the system. Later, same authors [KPD+14] proposed intelligent load balancing techniques for brownout-aware cloud services with multiple replica for improving the resilience of the services. Content reconfiguration takes into account only the system response time so that to prevent system instability in sudden workload burstiness. While the novelty of the approach is well understood, how the controller should be designed and implemented in massively virtualized cloud environment, has not been addressed. In n-tier applications, bottleneck for system performance could be different in each tier that could affect overall service time for different workload profile.

The work at [DLAdOJT15] proposed multi-level elasticity (*e.g.,* both in Software and Infrastructure) framework based on autonomic computing to improve the system performance by providing tactics associated to each layer and coordinating them by a local controller. While elasticity at the infrastruture level is very well understood by dynamically removing or adding resources, later requires a significant initiation time, for instance, VM start up time takes 150 seconds on average [MH12] and the dynamics of a cloud application can change faster than the resource activation time. Apart from that, resources are limited and can not be provisioned infinitely. Therefore, to tackle the problem, authors proposed horizontal and vertical scaling in application, while former refers to add/remove software components on the fly, latter refers to change the offering of existing software component. By doing this, systems resource requirement can be reduced for shorter period of time when activation of new resource is required but yet not ready to serve. Nonetheless, more client request can be accepted and served although the service may operate at a degraded level. An example of degraded solution could be: calculating and fetching 1 itinerary from google maps to clients instead of 3 or more itineraries. Although, the later work tackled adaptation of SaaS application under various criterion to improve availability and revenue, how modern cloud application can adapt with the proposal is yet to be validated with real life applications.

While, most of the work mentioned above discussed about self-adaptation techniques in SaaS layer for cloud application, they lack the anticipation of subsequesnt adaptation needs and effects to the system. Due to the reactiveness of the adaptation techniques, system can be triggered to perform a series of suboptimal adaptation, resulting inefficiencies. For example, when adaptations have latencies and requires some time to produce the effect on the system, using a proactive approach is necessary so that they have initiated before to impact effective immediately when it is needed. To this, authors at [AMCGRS15] proposed a latency aware proactive adaptation to deal with the uncertainties by using probabilistic model checking for adaptation decision. The approach uses a look-ahead horizon that predicts the near future states of environment to find the adaptation that maximizes the expected utility accumulated over the horizon in the context of the uncertainty of the environment. The utility is a function of both system and environmental state. To estimate

the environment, authors proposed markovian decision process (MDP) to capture the prediction constructing probability tree. Root of the tree represents the current state of the environment, each node presents a possible realization of environment and each children corresponds to realizations conditioned on parents. Moreover, their stated model can capture near future change in the system, thus it made easy to choose better adaptation tactics depending on the needs. Later, same authors [AMCGRS16] proposed how different non-conflicting tactics (*e.g.,* remove one software component and add one server) can be triggered simultaneosly so that system can transition from current to desired state. The challenge is to estimate how two types of tactics when applied together reacts to the system. For instance, removing software component can have immediate transition, whereas adding one server can make a delayed transition. Therefore, each of the tactics are associated with cost. Depending on the goal, the utility function can be maximized by choosing proper adaptation tactics.

Non-functional adaptation Most of the non-functional adaptation techniques relies on rule based reactive, pro-active and analytical model based approach. Usually, rule based reactive approaches[Ser14] [Mic14] are widely adopted in the public cloud and users typically specify upper and lower bounds of monitored metrics and based on the specification, resources scales in both ways. In contrast, pro-active approaches [YF13] [MBL+13] [KKJ11] are mostly predictive and try to formulate a plan before any occurance appears. Furthermore, analytical models [MHL+11] [PWAJ15] are build [PSZ+07] through dynamic measurement, evaluating correlation between parameters and correcting them on online.

Padala et al [PSZ+07] first addressed the problem of dynamically controlling resource allocations to individual components of multi-tier enterprise applications in a shared hosting environment. The authors proposed an adaptive integral controller based on resource utilization to keep the relative utilization under a threshold to maximize the resource utilization where application level QoS can be guaranteed. The controller algorithms were designed based on input-output models inferred from empirical data using a black-box approach. In concrete terms, system was modeled by varying the input in the operating region and by observing the corresponding output (see Figure 3.9) by building an analytical model. In contrast to queueing theory, which is widely used in computing system to aggregate statistical measure of a system, classical control theory can provide better run-time control over short time scales by piggy-backing fine-grained resource consumption, for example, CPU utilization and latency. Later, they designed arbiter controller that control dynamic resource allocation across multiple application tiers sharing the same infrastructure to keep throughput and response time under control. Therefore high resource utilization and performance was guaranteed. However, this work only considers CPU allocation and requires offline training for model building and clearly ignored how to manage other resource types and is categorized as reactive. Moreover, not all open source cloud management platform supports cpu entitlement techniques.

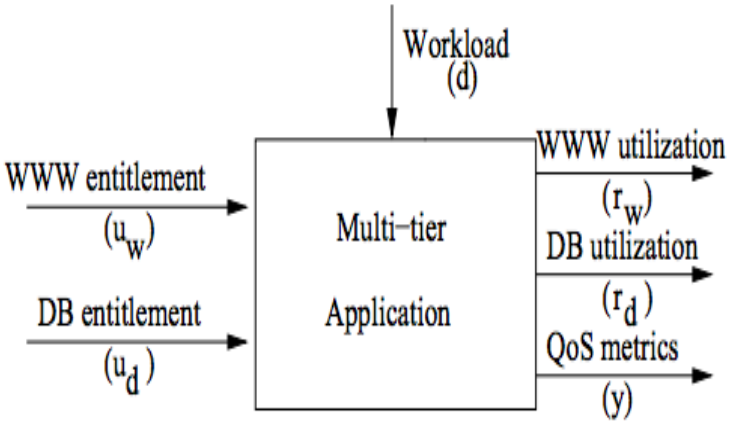


Figure 3.9: Input-Output model for a multi-tier application [PSZ+07]

In [KKJ11], authors introduced an online resource provisioning mechanism to improve performance management for session based enterprise application using limited lookahead control (LLC). This method is a pro-active model predictive control which try to maximize performance objective in a optimization problem over a predicted horizon, and periodically moves the horizon forward. Usually server uses remote procedure calls for client/server interaction and each client request results in the creation of a certain number of these calls. Authors used PID controllers to keep a setpoint which indicates the maximum number of clients the system can admit. By taking feedback from the system about number clients and closely tracking to the set point can be a realization point of triggering scaling decision on the infrastructure. Similar to this, [KCH09] [GDK+14] proposed pro-active approach by Kalman filtering to automatically learn the system parameters for each application, allowing it to proactively scale the infrastructure to meet performance guarantees.

Unlike using control theory, authors in [MHL+11] proposed a multi-modal controller that integrates adaptation decision from several models to determine best adaptation action *e.g.*, in terms of lowest adaptation cost while abiding performance level SLA. Their solution is based on empirical model based on measurement data from previous application runs so that, this model can suffice to be a starting point for a new application and by the time passes, knowledge based continous improvement can be done. The proposed controller can determine the number of instances required for each tier, when it is required. The solution is called integrative adaptation engine (IAE), which is consists of three main components; elastic application system, multi-modal controller and operational data store (ODS) as showed in Figure 3.10. During the run-time operation, the application can be monitored and operational data like resource utilization, number of instances per tier, throughput etc. are stored at ODS. In contrast, multi-modal controller sense SLO data via

sensor and matches the condition with ODS data, determines if system can be optimized or not, if yes, triggers proper adaptation decision via actuator. If a proper match is not found, horizontal scale model is used to determine the over or under utilized tier and then adaptation decision is performed. Similar to the goal, authors in [HGG+14], [KC14] proposed cost and QoS aware adaptation techniques in VM level for n-tier application.

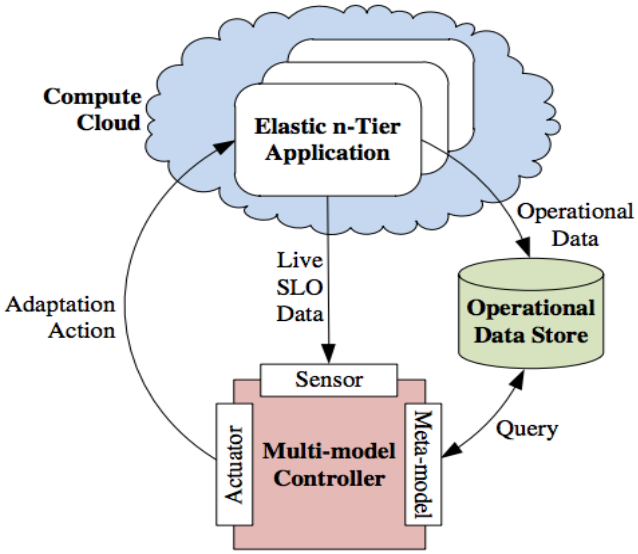


Figure 3.10: Integrative adaptation engine architecture [MHL+11]

While most of the non-functional adaptation techniques used in literature are mostly rule based [Ser14] [Mic14] using infrastructure and system level metrics, authors at [SN15] studied how effective autoscaling techniques could be using data generated by application itself. For instance, data streaming application like IBM Streams [CdCGdM+15] evaluates tweet sentiment in real-time. Their analysis shows that there exist a correlation between average sentiment of tweets and average volume of tweets. Therefore, by analysing intensity of the sentiment, upcoming tweeter burst can be alerted beforehand. To find a better solution depending on the application data, authors proposed two algorithm, namely load algorithm and appdata algorithm. The former technique is based on the expected time to process all current tweets versus the given SLA. For instance, each tweet must be processed withing 5 minutes. If the expected delay is above the SLA, more resources are allocated, otherwise resources are released and if the expected delay is below half the SLA. In contrast, the appdata algorithm analyzes the average sentiment score of the last minutes and compares it to the average sentiment of the previous minutes. If the sentiment score increases by a specified threshold, a predefined quantity of new CPUs is allocated. Results show that, resource requirements can be reduced upto 33%, while number of SLA violation can be also reduced by 95%. However, the approach is very fitted for data streaming

application.

The author [PWAJ15] proposed an efficient resource management of shared resources in virtualized environment based on nonlinear relationship between input and output parameters. Moreover, service provider can propose differentiated services in terms of performance *e.g.,* response time, quality of contents, etc. based on a defined business model *i.e.,* gold, silver and bronze. Depending on the business model, number of clients or workload can vary from one instance of application (granular to the VM level) to another. Therefore, hosting different class of applications of VM's in same physical machine makes it complex to gurantee different level of performance. Primarily, performance of a VM are non-linear related to the shared resources [PWCH11, WZS05]. Allocating more resources *e.g., cpu and memory* to a particular VM can degrade the performance of rest of the VM's. Therefore, a predefined linear model can be inadequate when system has to operate in a wide range of operating conditions with unpredictable workloads. Their solution successfully tackles the problem by designing a pre-input and post-input nonlinear compensator which takes feedback from the system to compensate these nonlinearities in the system.

Apart from these works, some of the research article tried to investigate power-performance trade-off. Rather, adapting the application, these works proposed to minimize the number of underlying resources used by application while acceptable performance can be provided. The authors in [HWY+10] introduced an online self-reconfiguration approach based on genetic algorithm to optimally reallocating VMs. Their approach conserve energy by effectively switching off unnecessary PMs. However, these approaches do not address how switching of PM's and migrating VM's can impact on modern interactive Cloud applications while targeted performance can be met. Additionally, authors in [CGS+14], presented StressCloud, which profiles the impact of system resource allocation strategies for different types of cloud tasks (*i.e.,* data and communication intensive tasks) to minimize energy consumption while performance requirement can be met without degradation.

3.4 Discussion

The section 3.3 presented a selection of relevant and popular work around the self-awareness of cloud application depending on green energy and performance. Furthermore, we define some attributes that classify the above mentioned works for better understanding the problem that we want to address:

- **Green energy awareness:** By the term, we mean that an application is aware of the variability of green energy and change it's execution time or makespan accordingly.

- **Cost optimization:** Most of the data center research are concentrated on how to reduce electricity related cost. Grid-tied energy cost can be reduced by opportunistically

using available green energy, whereas some efforts have been made to exploit location and price diversity for further reduction.

- **Performance guarantee via adaptation:** Performance is the foremost criteria for any cloud application but it can be varied depending of the nature of application. For example, abiding the deadline is a primary requirement for batch application. On the other hand, latency and availability is the key concern for interactive application.

From the work that takes into consideration of green energy awareness for batch kind of jobs, all of them [HGRN15], [DSFH15],[LOM15] use non-functional adaptation (*e.g.,* addition of VM, PM and VM migration etc.) The primary focus of these works are to meet the deadline while green energy usage can be maximized. However, the approach [LOM15] shows that, while proportion of green energy usage was maximized, total energy consumption was also increased. Therefore, not only the green energy needs to be consumed more, but also the total energy consumption should not exceed the acceptable level. Apart from that, [GBL+11], [GLN+12] not only tried to maximize the usage of green energy, but also to reduce the grid-tied brown energy cost. Furthermore, for the cost minimization metrics, two types of works are found in literature. One of the branch [WZL14], [TB15], [APKSG14], [GDXHJ14], [GZG+16] tried to route web requests across the data center to reduce electricty cost by exploiting the different prices in location diversity. Whereas, the other branch of works [CJX14], [LLW+11], [CHT12], [APKSG13] follows the renewable energy across the data center to take the advantage of available green energy on those location to reduce brown energy cost. Therefore, most of this works are based on traffic routing or require some form of non-functional adaptation for performance guarantees. In a way, these approaches tried to make an application to green energy adaptive by following the renewables. In literature, works related to interactive cloud application mostly emphasize to guarantee performance by doing architectural change in run-time to admit more requests to defend shortage of resources (functional adaptation), or by adding more resources to satisfy demand spikes (non-functional adaptation) or by formulating both of them. These adaptation policies actually opens the door for creating the green energy adaptivity to the interactive cloud application. However, the literature still lacks the strategy of how the interactive application can take advantage of available green energy to re-configure itself to the changing condition to reduce brown energy while traditional performance requirement can be met. Table 3.2 summarizes all the works we have discussed at Section 3.3.

In order to overcome the problem of creating green energy awareness in an interactive cloud application, our second part of the thesis in Chapter 6 presents a self-adaptive autoscaler architecture (thanks to autonomic computing) to enable smart usage of energy in an interactive application. Our idea is to transport the energy information to the application so that it can adapt itself based on the energy events to reduce brown energy consumption. We propose several application controllers that make trade-off between QoS, QoE, brown

and green energy consumption to acknowledge the idea of smartly adapting the application is possible with the energy information.

Table 3.2: Cloud Application

Application type	Approaches	Green energy awareness	Cost	Performance through adaptation	
				Functional	Non-functional
Batch	[HGRN15], [DSFH15],[LOM15]	✓	✗	✗	✓
	[ZWW11], [GLN+12],[LCB+12], [CHT12]	✓	✓	✗	✓
	[GBL+11], [CJX14],[LLW+11], [APKSG13]	✓	✓	✗	✗
	[WZL14], [TB15],[GDXHJ14]	✗	✓	✗	✗
	[APKSG14], [GZG+16]	✗	✓	✗	✓
Interactive	[AB99], [ASB02],[PDPBG10], [CFF+11], [KPD+14]	✗	✗	✓	✗
	[CIAP12], [KMAHR14]	✗	✓	✓	✗
	[DLAdOJT15], [AMCGRS15]	✗	✓	✓	✓
	[AMCGRS16]	✗	✗	✓	✓
	[PSZ+07], [KKJ11],[GDK+14], [PWAJ15]	✗	✗	✗	✓
	[MHL+11], [HGG+14], [KC14], [SN15], [HWY+10], [CGS+14]	✗	✓	✗	✓

Part II

Contribution

Chapter 4

Cloud energy broker: Green energy planning for data center

Demand for Green services is increasing considerably as people are getting more environmental conscious to build a sustainable society. Therefore, enterprise and clients want to host their services or applications towards *Greener Cloud Environment* that offered by the Infrastructure-as-a-Service (IaaS) provider. To build a greener cloud environment around data center, maximum energy efficiency and minimum environmental impact (*i.e.,* lower carbon footprint) are the foremost criteria. To this, several energy efficient techniques for hardware and software systems have been proposed in the literature that are widely adopted and practiced. In contrast, data center's energy management in the presence of implicit and explicit sources of green energy that can facilitate to reduce carbon footprint is still in its infancy, but gaining a lot of attention lately. The main challenge for an IaaS provider is to determine the best trade-off between its profit while using green energy with a budget constraint and contracted Service Level Agreement (SLA) with Service-as-a-Service (SaaS) and Energy provider. On the other hand, for providing Green computing services to the SaaS provider or client, strong SLA needs to be addressed. Therefore, in this chapter we explain different level of Service Level Objective (SLO) in each Cloud layers to realize how cross-layer SLA can be contracted in the presence of green energy in Cloud computing environment. Furthermore, we propose a *Cloud energy broker*, which can adjust the availability and price combination to buy Green energy dynamically from the energy market in advance to make a data center green based on contracted SLA. Later, validation of the energy broker is provided to show it can successfully keeps the best trade-off between energy availability and budget constraint. Moreover, this chapter presents the planning phase of green energy management for data center.

4.1 Context and Motivation

In response to the growing demand for Internet and Cloud computing services, large companies such as Amazon, IBM, Google, Yahoo!, Microsoft etc. responded greatly by making their own Cloud platforms and datacenters. It is obvious that data centers consume enormous power that can lead to negative environmental implications (e.g., emission of several million tons of CO2 and global warming) in its life span, which is a serious concern for society and academia researchers in recent years [HH13]. Similar to other large consumers of power, data centers find themselves increasingly pressured either by legislation or by public opinion to find options to reduce their carbon footprint. Therefore, demands for green products and services are ever increasing. In response, using green energy in the data center is one of the best ways to address this issue even though green energy sources are very intermittent in nature and generally incurs higher cost to produce energy.

Green energy driven SLA. Due to the dynamic nature of the Cloud, SLA between consumers and providers emerge as a key aspect and SLA enforcement becomes an important challenge. Today's research is more concentrated on Workload-driven SLA rather than Power-driven SLA and Green power-driven SLA. Usually, Workload-driven SLA depends on end-users criteria such as availability, response time, throughput, etc. In contrast, Power-driven approach implies, shifting or scheduling the deferrable workloads to the time period when the price of electricity is lower or migrating workloads to the different region (data center) where the electricity price is cheaper than the origin while respecting the deadline. On the contrary, Green power-driven SLA can be realized as: end-users or SaaS providers shift their workloads in a renewable/green energy powered data center having an agreement with IaaS provider that some portion of their workload should run in a greener environment. To do that, we should re-visit and re-design existing SLA framework to propose a new class of SLA based on green energy.

Underlying energy sources for data center. Existing literature does not provide a clear idea about the advantages and disadvantages of different integration option of green energy sources in data centers. While on-site and off-site renewable generation models are explicitly involved with data center to offset their carbon footprint reduction goal, some implicit model e.g., Renewable Energy Certificate (REC) and Power purchasing agreement (PPA), have created lots of attention to the data center owners or Cloud providers. In REC market, there exists several green energy providers who will produce energy and feed to the grid. As green energy sources are very intermittent in nature, the green energy-feeding price would be very different from one to another provider depending on the location of site, availability of sources (Wind speed, solar irradiation etc.) and capacity factor of the plant. Committing to a single energy provider might result unavailability of required green energy requirement for certain time frame thus ensuring certain percentage of green energy availability in data center can not be met. On the other hand, when the generation of green energy is lowest due to weather or maintenance work in the plant, the price

of energy also might go beyond the acceptable limit. Thus, providing contracted Green computing services to SaaS providers or end-users become extremely difficult for a IaaS provider. Therefore, an efficient solution is necessary that can ensure maximum green energy availability by exploiting different energy integration option while budget constraint is respected.

4.2 Energy procurement and Integration

To propose green computing services powered by renewable/green energy, first we have to investigate the different green energy integration options and their advantages as well as disadvantages and procurement strategies. From Figure 4.1, it can be shown that, the energy layer of the IaaS consists of a single Grid where several *Green Energy-as-a-Service* (GEaaS) providers from REC market and green energy provider from spot market are connected. Additionally, on-site green energy plant can be associated to the data center internally or externally through the same Grid. Following are the different green energy integration opportunities for a data center.

- *On-site green energy*: Due to the growing demand of green services, most of today's green data centers adopted on-site green energy plant e.g., wind turbine, solar panel to meet the green energy demand. Nevertheless, the perfect place for constructing a green energy plant might not have the true potential to build a data center due the intermittent nature of the renewable sources. But having a small-scale renewable plant always gives the advantage to incorporate green energy to the data center to fulfill at least the partial green energy demand if there is not sufficient amount of energy in the REC or Spot green energy market.

- *Off-site green energy*: Incorporating off-site green energy to data center is an alternative option, as the best location for producing green energy does not always have the best potential to build a data center. Transporting the off-site energy is arduous as wheeling charge imposed by the Grid might be more than the expectation and power losses through transmission lines are inevitable. Thus it is not suitable or preferable option for small-scale data center.

- *REC market*: While on-site and off-site renewable generation models are explicitly involved with data center to offset their carbon footprint reduction goal, some implicit model e.g., Renewable Energy Certificate (REC) and Power purchasing agreement (PPA), have created lots of attention to the data center owners or Cloud providers. REC, known as green certificate in Europe, is a tradable commodity proving that electricity generated using renewable sources. [1] Therefore, purchasing of a green

[1] http://www.recmarket.eu/

certificate equals to purchasing a claim that the certificate owner consumed energy from the renewable portion of the whole energy grid.

- *Spot green energy market*: Usually spot market posses lesser amount of energy than regular energy or electricity market and price tends to be higher than traditional or different non-flat tariffs. Spot market is very important for consumers like IaaS provider, if the real-time energy/power demand is excessive than the forecasted demand. Moreover, the actual demand cannot be known accurately in advance and any forecasting technique provides at least some error statistics. So, for fulfillment of SLA based on green energy, IaaS provider needs to purchase green energy from spot market if it is required.

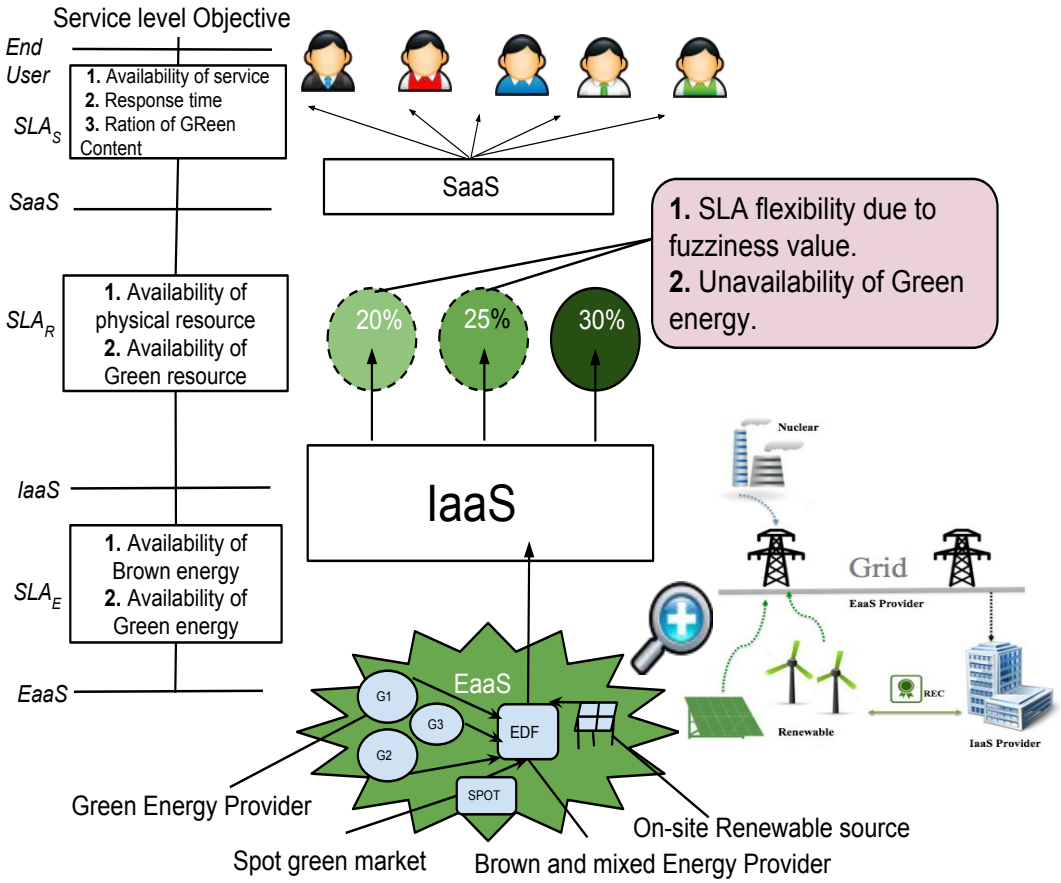


Figure 4.1: Cross-layers SLA

4.3 SLAs in different layers

To establish green energy dependent cross-layer SLA, it is imperative to define different actors in different cloud layers. Therefore, the objective of this section is to present the SLA dependencies in a Cloud cross-layers architecture. First, we present the actors involved in different layers. Later, we describe Service Level Objectives (SLO's) associated with each layer.

4.3.1 Actors

The Cloud architecture is usually composed of several XaaS layers and SLAs are characterized at various levels in this stack to ensure the expected QoS for different stakeholders. As shown in Figure 4.1, an End-User is a client of the SaaS provider, which is itself a client of the IaaS provider and as well as for Energy-as-a-Service (EaaS) provider.

In the REC market, GEaaS providers produce green energy and feed to the Power/-Electrical grid but sell their green energy or green energy credits in a wholesale market to consumers (*i.e.*, IaaS provider) for direct purchase. Even though IaaS provider consumes energy from the Power/Electrical Grid, they have to pay directly to GEaaS providers for their consumption of certain portion of the green energy which has been contracted with respective GEaaS provider. In a smart grid and smart city environment, Grid monitoring organization[2] is responsible for monitoring the energy production added to the grid and consumption of different small to large power consumers. Since the SLA has to be contracted between IaaS provider and GEaaS providers, the Grid monitoring infrastructure mentioned in Figure 4.2 is considered as supporting part (or third actor) to monitor/validate SLA between IaaS and GEaaS providers. Except for the End-User, any Cloud layer plays a provider-consumer role: it is a provider for the upper layers and a consumer for the lower layers. Its main challenge is to maintain consumer's satisfaction facing to the demand variations while minimizing the service costs due to resources fees and SLA penalties (in case of violation).

4.3.2 SLAs

The Figure 4.1 presents examples of Service Level Objectives (SLO's) that apply at three different Cloud levels; between the End-User and the SaaS, the SaaS and the IaaS, or the IaaS and the EaaS:

- SLA_S (End-user – SaaS provider): Service Response Time, Service Availability.
- SLA_R (SaaS provider – IaaS provider): Resource Availability, Green Resource (percentage of used green resource).

[2]For example in France - http://www.enedis.fr/

- SLA_E (IaaS provider − EaaS Provider): Brown energy Availability, Green energy Availability.

The Listing 4.1 presents an example of code in CSLA[KADOJDL14], a SLA language to finely express SLA and address SLA violations in the context of Cloud services. CSLA allows defining SLA in any language (e.g., XML, Java); we use XML as a representation format for the sake of simplicity. This code describes the guarantee terms and penalties for SLA between a IaaS provider and its customer (SaaS provider).

In this example, we focus only on one SLO about the percentage of green resource (lines 1-5). The SLO states that at least 30% of green resource should be guaranteed, with confidence, fuzziness and percentage fuzziness of 83.33%, 5 and 30%, respectively. These CSLA features (confidence, fuzziness) have been introduced to deal with QoS uncertainty in unpredictable Cloud environment [KL12]. In concrete terms, it means that the percentage of green resource measured within an observation period may be i) lower than 25% in 16.67% of the observation periods, ii) between 25% and 30% in 24.99% (83.33% of 30%) of the observation periods and iii) greater or equal to 30% in 58.33%. A violation of the *GreenResource* SLO implies a penalty that depends on the green percentage not respected (lines 6-13). For each penalty, a procedure (line 9) indicates the actor in charge of the violation notification (e.g., provider), the notification method (e.g., email) and the notification period (e.g., 7 days).

Listing 4.1: CSLA example.

```
1  <csla:terms>
2  <csla:objective id="GreenResourceSLO" actor="provider">
3  <csla:expression metric="Gr" comparator="gt" threshold="30" unit="\%" monitoring="Mon−1" Confidence="83,33"
       fuzziness−value="5" fuzziness−percentage="30"/>
4  </csla:objective>
5  </csla:terms>
6  <csla:penalties>
7  <csla:Penalty id="p−Gr" objective="GreenResourceSLO" condition="violation" obligation="provider">
8  <csla:Function ratio="0,5" variable="GreenPercentage" unit="\%">
9  <csla:Procedure actor="provider" notificationMethod="e−mail" notificationPeriod="7 days">
10 <csla:violationDescription/>
11 </csla:Procedure>
12 </csla:Penalty>
13 </csla:penalties>
```

4.4 Components of Cloud energy Broker

As mentioned earlier, IaaS provider needs to determine the best trade-offs between costs associated with different GEaaS providers with the available amount of green energy needed to satisfy contracted SLA. In order to address this issue, we provide a *Cloud energy broker*, which can adjust the availability and price combination to buy Green energy dynamically from the market in advance to make a data center partially green. We also have taken a realistic consideration that Green energy providers can publish a day ahead green

energy generation and price per hour (see the left side of Figure 4.2), which is a common practice at European electricity and energy market along with smart-grid environment. For the sake of simplicity, we have shown only one GEaaS provider in Figure 4.2. The proposed cloud energy broker is composed of several components that interacts with each other. Figure 4.2 provides the main components of our broker framework.

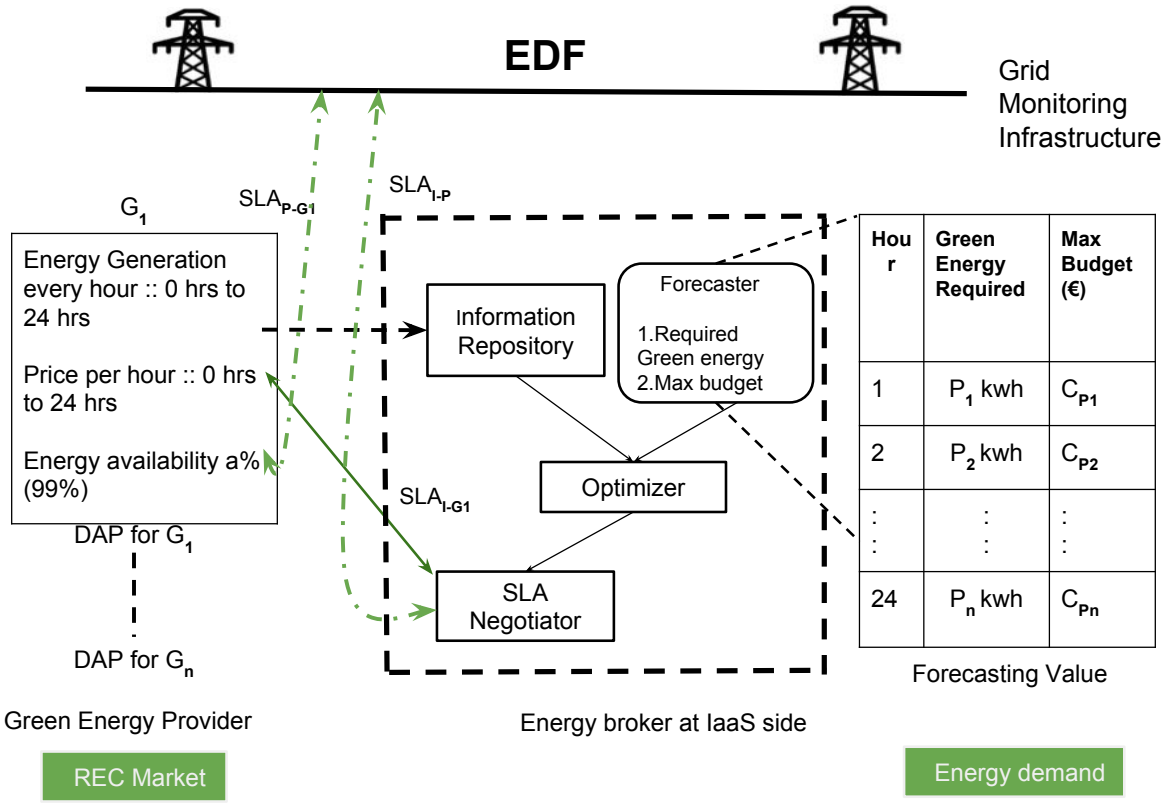


Figure 4.2: Top level view of the framework

- *Information repository*: this component stores Day Ahead Pricing (DAP) information published by different GEaaS providers. The information is updated instantaneously if any change has been made at DAP information of GEaaS providers, otherwise the information is updated periodically in a given time frame.
- *Forecaster*: the amount of Green energy required for IaaS can be forecasted for the next few minutes (short time forecasting) or the next few hours (long time forecasting)

based on k days of energy usage. Once the requirement of Green energy is forecasted, the component can calculate the maximum Green power budget from the history or from IaaS provider's power budget information.

- *Optimizer*: both Information repository and Forecaster forward their information to the Optimizer component. Therefore, the Optimizer provides pareto optimal solution for dynamically selecting GEaaS provider based on respective information for each time interval, for example 1 hour.

- *SLA Negotiator*: after selecting desired GEaaS provider/providers for each time frame (1 hour), this component establishes a SLA contract between IaaS and GEaaS provider. In addition, the SLA negotiator also makes a SLA contract, denoted by SLA_{I-P} with the Grid infrastructure for monitoring the violation of contract in the case where Green energy is not delivered to the Grid. On the other hand, Grid monitoring infrastructure establishes SLA denoted by SLA_{P-G1}, to monitor if the energy providers have fulfilled their commitment of adding desired/contracted amount of green energy or not. Therefore, to validate the SLA_{I-G1} between IaaS provider and green energy provider, two supporting agreement or SLA, named SLA_{I-P} and SLA_{P-G1} are required.

4.5 Planning phase and life cycle

The selection of a GEaaS Providers can be abstracted as a succession of operations in a planning phase (see Figure 4.3). The complete lifecycle includes both IaaS and GEaaS providers information: forecasting power demand of IaaS, day ahead pricing (DAP) data of GEaaS providers, selection of best GEaaS provider or providers, buying dynamically Green energy from GEaaS providers. Moreover, planning framework is divided into two time frames: hourly and daily (*e.g.,* 'm' hours,'k' days). The first phase ends with step 4, where buying Green energy is dynamically done hourly. Once GEaaS providers update DAP information, a new schedule is initiated, thus concludes the process for 'm' hours. In addition, the second phase resolves the process by step 5 for 'k' days. Therefore, forecasting power demand and selecting energy providers are the key concerns while designing an energy broker where latter part requires optimization based on forecasted data and published pricing information from energy providers.

Monitoring and Forecasting: Predicting power demand in Cloud computing environment is very arduous as ratio of power consumption at different infrastructure (e.g. servers, cooling, lighting etc.) level are very divergent. Therefore, using Power Usage Effectiveness (PUE) helps to get better understanding about power demand of a data center. For a data center, PUE is defined as the ratio of the data center's total power consumption to the data center's power consumption at the computer servers, databases and networks [GMR13]. Hence, if we can measure the power consumption at server level, it becomes easy

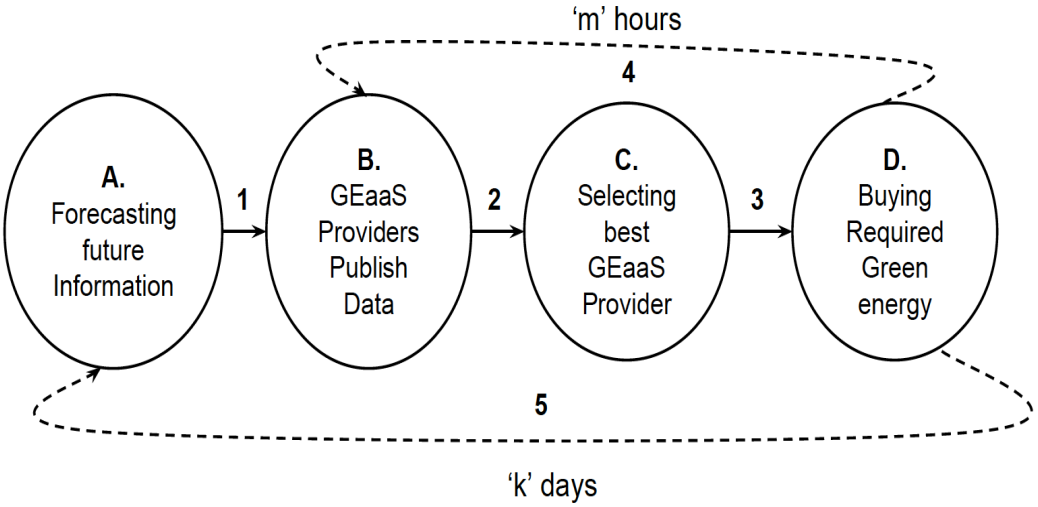


Figure 4.3: Planning life-cycle

to calculate the total power consumption of a data center for certain time frame. As CPU consumes majority of the power compared to memory in server level, in our investigation we ignore the power consumption by memory in the power model. Furthermore, future demand of power consumption can be generated by using efficient forecasting method. The output of the forecasting phase is E, where E represents the requirement of green energy for next 'm' hours.

Optimizing: The goal of our optimization framework is to the find optimal amount of energy from GEaaS provider or composition of best GEaaS providers while respecting the budget. We address our optimization problem as Constraint programming (CP) [RBW06], since CP accepts any type of relations to formulate constraints consisting of linear inequalities. So, variable X_i represents the amount of Green energy required by IaaS provider for each time interval from G_i (where, $i \in [1, ..., n]$) provider, whereas Domain $D(X_i)$ demonstrates the DAP information published by GEaaS providers.

- Variable: $X = \{X_i \mid i \in [1, ..., n]\}$
- Domain: $D(X_i) = \{1, ..., e_i\}, \forall i \in [1, ..., n]$

Therefore, we introduce our objective function which tries to maximize both the amount and the availability of Green energy to meet the exact Green energy requirement.

$$Maximize \quad : \quad (\sum_{i=1}^{n} X_i . \prod_{i=1}^{n} Av_i) \tag{4.1}$$

where, Av_i symbolizes the availability of X_i.

$$Subject\ \ to \quad \sum_{i=1}^{n} C_i \leq B^{max}, \quad B^{max} \in \Re_+ \tag{4.2}$$

IaaS provider requires to have upper bound of budget for each interval to buy Green energy which is stated at constraint (4.2) as B^{max}, where B^{max} is computed by (E)*($St.Price$) and C_i represents the cost for buying green Energy from provider G_i. The term (E) and ($St.Price$) represent the required Green energy and average green energy price from historical window respectively.

4.6 Evaluation

This section presents the results obtained from an experimental scenario used to evaluate the proposed broker. The objective is to show a real utilization case of the forecaster and the optimizer.

Table 4.1: Power consumption by the selected servers at different load levels in Watt

Servers	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Dell Inc PowerEdge M620	688	1151	1322	1494	1671	1848	2061	2289	2499	2765	3239
IBM NeXtScale nx360 M4	550	873	999	1123	1251	1380	1525	1673	1887	2116	2404

4.6.1 Experimental Testbed

We consider a data center which has an average PUE of 1.77. Although some of the state-of-the-art techniques claim to have reduced this value closer to 1.20, still most of today's data center have higher PUE values than 1.7 [YM13]. Therefore, for transforming CPU utilization to power consumption, we traced CPU utilization for 7 days of 30 blade servers from PLANETLAB [PP06] where CPU Utilization has been traced for 500 different servers from across the world. As, building precise analytical models for modeling power consumption by modern multi-core CPUs makes a complex research problem, instead of using an analytical model of power consumption by a server, we utilize real data on power consumption provided by the results of the SPECpower benchmark [3]. Theoretically, researchers assume the increment of power consumption is linear to the increment of CPU load, whereas practically the power consumption in a server is not linear and increase significantly beyond 80% CPU utilization.

[3] http://www.spec.org/power_ssj2008/

4.6.2 Forecaster Evaluation

We have selected two blade servers configuration with multi-core CPUs published in November 2013: Dell Inc. PowerEdge M620 (Intel Xeon E5-2660,8coresX2.2 GHz,64 GB), and IBM NeXtScale nx360 M4, (2 Intel Xeon E5-2600,10 coresX2.2 GHz,256 GB). The configuration and power consumption characteristics of the selected servers are shown in Table 4.1. In addition, we use OpenForecast [4] to forecast power demand for next 24 hours based on traced last 7 days power consumption. Single variable polynomial regression, Simple exponential smoothing and Double exponential smoothing method are used as forecasting method. Figure 4.4(a) and 4.4(b) present 7 days traced CPU utilization and transformed power consumption from CPU utilization for 7 days respectively. On the other hand, Figure 4.4(c) shows power consumption prediction for next 24 hours.

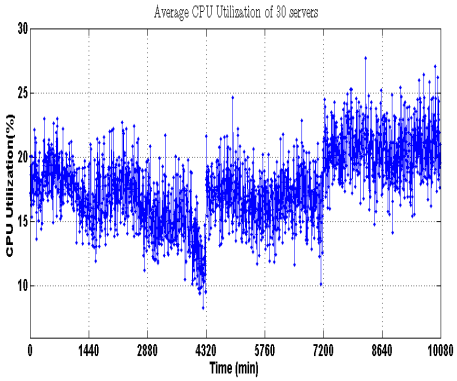
In smart city environment, the regulation authority could enforce large power consumers like data centers to have certain percentage of green energy in their data center. On the other hand, IaaS providers can set their own sustainability goal to keep certain percentage of green energy in data center to propose green services. For the evaluation purpose, we consider that IaaS provider's goal is to make the data center implicitly 30% green. Therefore, we scale down the power requirement demand to 30% which is shown in Figure 4.4(d).

4.6.3 Optimizer Evaluation

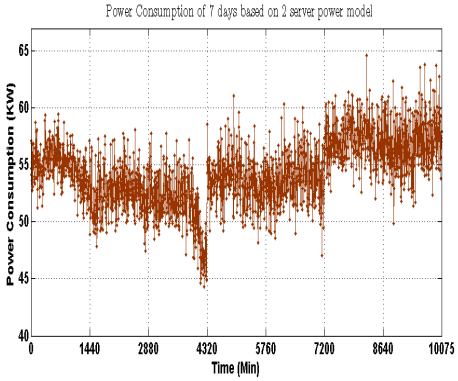
We consider 4 GEaaS providers exist in REC market for the purpose of our evaluation but it can be extended to more providers. As demonstrated in Figure 4.2, every GEaas provider has different level of availability of energy in kwh over time which is published at DAP information. The level of availability differs for various reasons including different wind speed over time, unavailability of cut-in wind speed, different solar irradiation over time and the capacity factor of the plants. Furthermore, some providers might use more than one or different sources to produce green energy, which also results different level of energy generation. Figure 4.5 shows the energy distribution by 4 GEaaS provider which is synthetacally created to validate the result. Using Riemann sum, we calculate the green energy consumption demand from Figure 4.4(d), as the billing or cost for consumption is always calculated over energy consumption in kwh rather than power consumption in kw. Furhtermore, for the ease of readability green energy demand is placed at Figure 4.5.

Finding market prices of each kwh produced by green sources are extremely difficult as most of the today's wind or solar power infrastructure or plants receive enormous incentives either from government or from different policy making organizations. Hence, to model a realistic price for energy of different GEaaS providers and energy purchasing budget for IaaS providers, we investigate information of CAPEX-OPEX, levelized cost, fixed O&M cost, variable O&M cost of different sources of energy (e.g.; Nuclear, Wind,

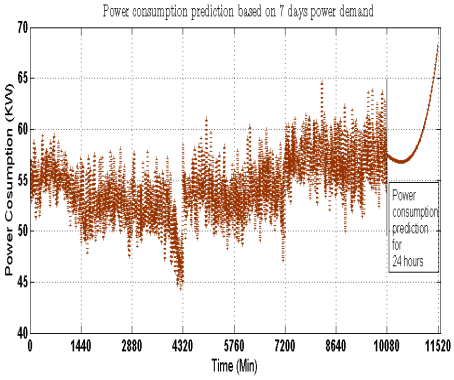
[4] http://www.stevengould.org/software/openforecast/index.shtml



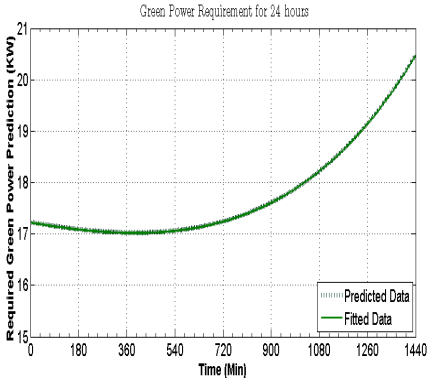
(a) CPU Utilization



(b) Power Consumption



(c) Predicted Power



(d) Green Power Prediction

Figure 4.4: From CPU utilization to Green Power Prediction

Solar, Hydro etc) [5] and find that the ratio of energy consumption cost between nuclear and green energy is *1:1.68* approximately. Therefore, we consider, the price of green energy sold by GEaaS providers will be around .19 - .25 cents/kwh while the price of Nuclear or mixed energy provided by EDF [6] is .13 cents/kwh.

In our experiment, we compare our optimization framework with two greedy approaches based on availability and cost. The first one tries to find the GEaaS providers such a way that it can satisfy the near optimal green energy demand whereas second ones seeks to select certain GEaaS providers which offer lower cost for selling their energy.

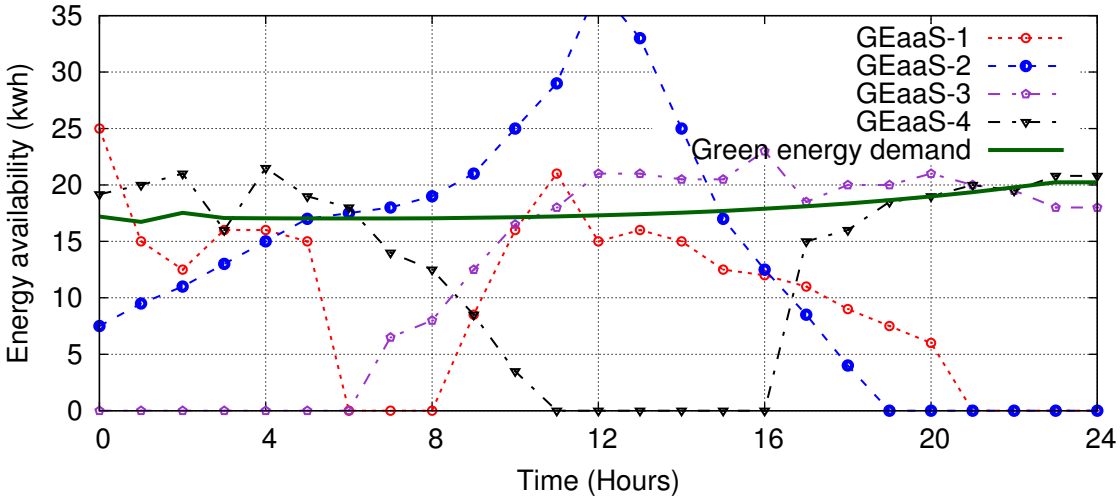


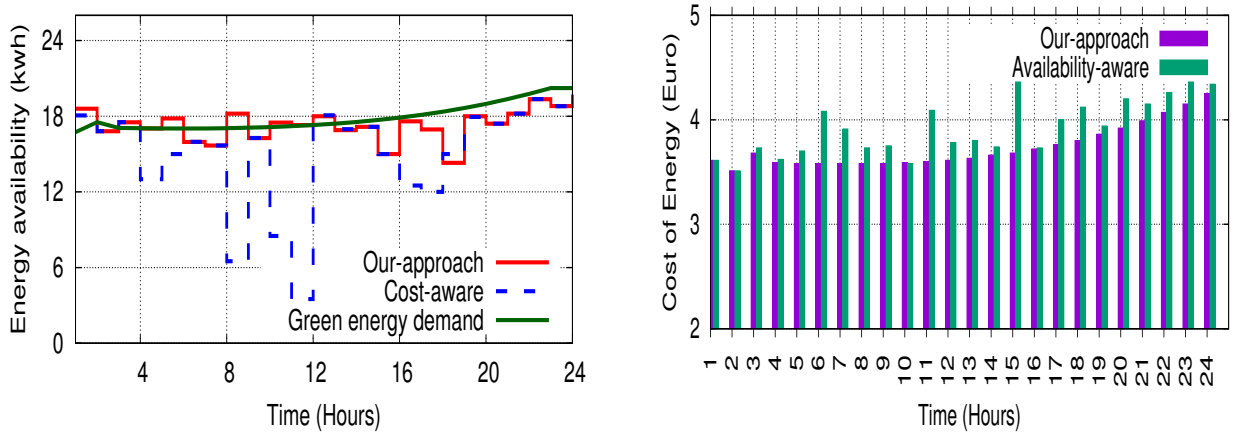
Figure 4.5: Energy production by different GEaaS providers

Results: Figure 4.6(a) shows the comparison between a cost aware greedy approach and our optimization approach to meet the forecasted green energy demand. While cost aware greedy approach fails to meet the energy demand by 14%, our optimization framework performs better by providing 98% of the total demanded green energy within the green energy budget of IaaS provider. Furthermore, availability aware greedy approach incurs 5% more cost than the green energy budget of IaaS provider, while our approach follows the budget strictly and fails to provide only 2% of demanded Green energy, which is showed in Figure 4.6(b).

4.7 Discussion

This chapter presented some insights into how green energy can be procured and added to a data center in the presence of different implicit and explicit green energy integration

[5] http://www.eia.gov/forecasts/aeo/pdf/electricity_generation.pdf
[6] http://entreprises.edf.com/entreprises-45638.html



(a) Cost aware

(b) Availability aware

Figure 4.6: Our approach vs Cost aware vs Availability aware

options. Later, how cross-layer SLA can be established in the existence of green energy is also discussed. Furthermore, a *Cloud Energy Broker* was proposed to tackle the problem of ensuring green energy availability under a fixed budget constraint to meet enforced green energy percentage by smart city authority or contracted green energy aware SLA between IaaS and SaaS provider, if there is any. We envision that, any green energy management strategy for data center requires a planning phase and a real-time execution phase. Since the actual energy demand can not be known accurately in advance and any forecasting technique provides at least some error statistics, real-time execution phase becomes very critical. In today's internet data centers, demand may vary rapidly *e.g.,* workload surge, which can not be anticipated precedent to the event. Thus shortage of green energy may occur. On the other hand, the periodic or "on and off" workload is characterized by periods of relatively high activity distributed with periods of little or no activity [7]. Due to the periods having less to zero activity, demand for green energy can decrease. Hence, green energy which is purchased beforehand can goes to waste. Considering the aforementioned occurrences, an efficient strategy needs to be devised for real-time execution phase. The latter chapter investigates and aims at solving the phenomena by introducing *Virtualization of Green Energy* for the data center.

[7] http://support.rackspace.com/white-paper/cloud-economics/

Chapter 5

Virtualization of green energy: Better managing the energy in datacenter

Defining and establishing *Green SLA* between IaaS and SaaS provider based on green energy is missing from the literature. Therefore, the main challenge for IaaS provider is to manage Green SLAs with their customers while satisfying their business objectives, such as maximizing profits by lowering expenditure for green energy. Aside from incentives from government and private organizations, cost for producing green energy and prices in the market tends to be higher than brown energy. Since, Green SLA needs to be proposed based on the presence of green energy, the intermittent nature of renewable sources makes it difficult to be achieved. In response, this chapter presents a scheme for real-time green energy management in the presence of explicit and implicit integration of green energy in data center. More specifically we propose three contributions: i) we introduce the concept of virtualization of green energy to address the uncertainty of green energy availability, ii) we extend the Cloud Service Level Agreement (CSLA) language to support Green SLA by introducing two new threshold parameters and iii) we introduce greenSLA algorithm which leverages the concept of virtualization of green energy to provide per interval specific Green SLA. Experiments were conducted with real workload profile from PlanetLab and server power model from SPECpower to demonstrate that, Green SLA can be successfully established and satisfied without incurring higher cost. Later we conclude the chapter by analyzing the limitations of our approach and what is the potential step to overcome the limitations.

5.1 Context and Motivation

While the proliferation of Cloud services have greatly impacted our society, how green are these services is yet to be answered. Although, demand escalation for green services has grown due to societal awareness, the approaches to provide green services and establish

Green SLAs remain oblivious for cloud or infrastructure providers. To provide green computing services, efficient green energy management technique in the presence of implicit and explicit integration of green energy sources is necessary. Following are our further arguments, which are the state-of-the-art problem to achieve an efficient strategy to manage green energy in the data center.

Drawbacks of energy storage. As renewable power sources are very intermittent in nature, hence predicting the amount of green energy production ahead of real time might demonstrate greater error statistics in data center. Nonetheless, excessive production of green energy can go to waste and sometimes might imbalance the Grid if the production becomes greater than the capacity. The later case does not apply if the data center has a small-scale renewable source e.g., wind turbine, solar plant. One way to overcome the challenge is to use energy storage or battery to store this superfluous green energy which can be discharged later for peak shaving of data center power demand or for fulfillment of *Green energy based SLA* between IaaS and SaaS providers when green energy is needed but not available. Energy storage incurs additional costs to data centers cap-ex and op-ex, hence it is not an attractive solution for small-scale data centers. Moreover, storages have finite capacities to recharge energy and their lifetime is a decreasing function of depth-of-discharge (DoD) and charge/discharge cycles [DLJ+13]. Usually DoD refers, how much energy the battery has delivered.[1] Therefore, if the production of green energy is above the capacity of storage, remaining energy goes to waste. Even the state of the art batteries have 80-85% [UUJNS11] efficiency on charging and discharging capabilities, which implies 28-36% loss of energy.

Then, how to manage unavailability of green energy at run-time if the storage approach has several drawbacks and the on-site green energy or spot market productions are insufficient?

Green SLA based on Green energy. While in the literature, reducing energy consumption per job (*e.g.*, at IaaS layer) is a primary concern to propose Green SLA [BKT12], we envisage that, Green SLA can only be proposed and established if the service is hosted in a green energy powered data center. To establish Green SLA, we should follow the bottom-to-top approach in cloud computing layer *e.g., EaaS* → *IaaS* → *SaaS*. So that, each of the layer can contribute their effort to reduce carbon footprint globally. Apart from that, any service providers are yet to propose Service Level Agreement (SLA) based on green energy availability with their infrastructure. Haque et al. [HLG+13] first proposed Green SLA based on green energy availability from on-site renewable plant where environmental conscious clients can ask for differentiated green services with varied green energy requirement. However, providing green service can be impossible or no additional green service request

[1] It is not recommended to fully discharge batteries to 100%, otherwise it would shorten the life-cycle of batteries.

will be entertained when green energy is not available, which quantifies that, it is not possible to have any formal contracts between SaaS providers/clients and IaaS providers. Hence, real-time energy management becomes very important not only to ensure the availability of green energy in the data center but also to validate Green SLA.

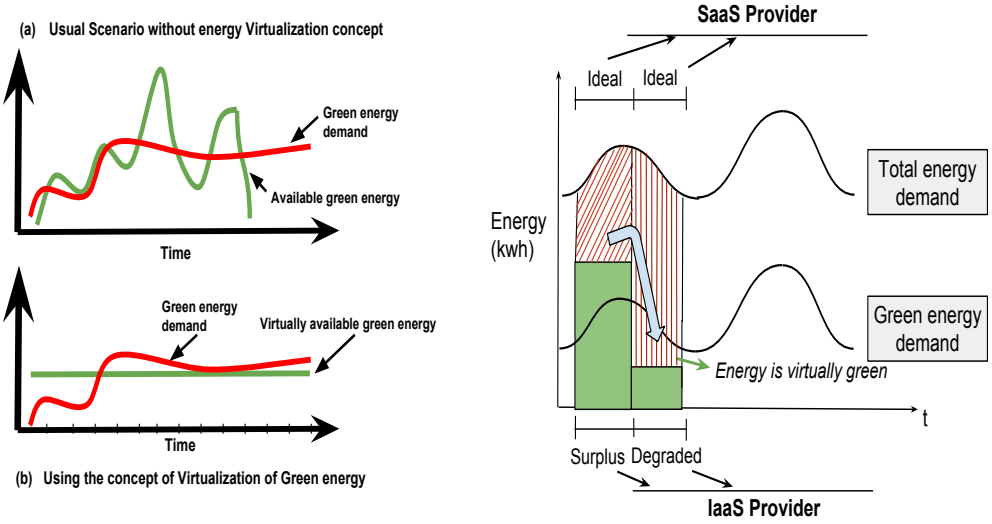
5.2 Proposed solution

Due to two time-scale green energy market (REC and Spot) and aperiodic spikes of workload we have divided our solution in two phase named planning and run-time phase. In previous chapter (Chapter 4 : Section 4.4), we proposed a *Cloud energy broker*, which can adjust the availability and price combination to buy green energy dynamically 24 hours ahead from the REC market to make data center green for a specific (as a example 30%) portion. To address the priorly mentioned green energy management problem in run-time phase, we introduce the concept of Virtualization of green energy and *Green SLA* based on the availability of green energy.

5.2.1 Virtualization of green energy

The energy can be virtually green for a specific period of time if abundance of green energy is available aperiodically in shorter time interval along with the deficit of green energy in rest of the time frame. Concretely, when the availability of green energy is more than demand, we use the whole portion of available green energy but characterize the interval as surplus interval. When green energy is insufficient to meet the demand, we nullify the degraded interval with the surplus interval. We use the term *virtualization* because we nullify a degraded interval (lack of green energy) with a surplus interval (excessive green energy than demand), but from the client's or SaaS provider's perspective, they realize both the intervals as ideal interval (when supply meet the demand), though the green energy was not present instantaneously rather present virtually. Figure 5.1 shows the visualization of our approach.

Therefore, our proposal is to smooth out the differences between deficit and surplus of green energy production during a certain time window with the objective to obtain an summation narrowly superior to a certain threshold, which we refer as *Virtualization of green energy*. This way we ensure that each watt of green energy is used in the data center. Interestingly, this concept does not increase the total energy consumption rather increases the *greenness of energy* used in data center. In this way *real* energy storage is not needed and neither of the portion of green energy is wasted. Furthermore, total expenditure for energy purchasing can be reduced as no energy goes to waste and additional cost for using storage is not needed. Obviously, *Green SLA* between IaaS and SaaS providers can be fulfilled if the time-slot length is adapted. For example, if IaaS provider has established a SLA to SaaS provider to have some portion of green energy available for each time slot e.g., T=30/60



(a) Effect of green energy virtualization (b) Validation of virtualization of energy

Figure 5.1: Green energy virtualization concept

minutes, it is possible to satisfy the contract by using virtualization concept of green energy, which is elaborately explained in the next subsection.

5.2.2 Extension of CSLA to support virtualization of green energy

We extend CSLA language to support the *Green SLA* by taking the advantage of the concept of virtualization of green energy. In order to evaluate an objective (SLO), an initial evaluation enables to classify the interval as ideal (i.e., threshold is respected), degraded (i.e., threshold is respected using fuzziness margin) or inadequate (i.e., threshold is not respected even with fuzziness margin). We distinguish two concepts: (i) per-interval evaluation, in which the evaluation is performed at the end of each interval; (ii) per-request evaluation, in which the objective is evaluated for each request. For the sake of virtualization of green energy (*e.g.*, since this concept is based on a certain time frame), we consider per-interval evaluation. A final evaluation, at the end of the time window, allows one to verify an objective (SLO) by applying the fuzziness and confidence percentages to the initial evaluation. The final evaluation enables the identification of non-accepted/accepted degradation and inadequate cases, that is, that will/will not result penalties. In other words, the final evaluation absorbs or notifies the violations.

To propose and establish Green SLA, first, we define an SLO by using two thresholds $threshold_{min}$ and $threshold_{max}$ (see Figure 5.2). Secondly, beyond the $threshold_{max}$, we consider the intervals as surplus meaning that, excessive green energy was present in that interval.

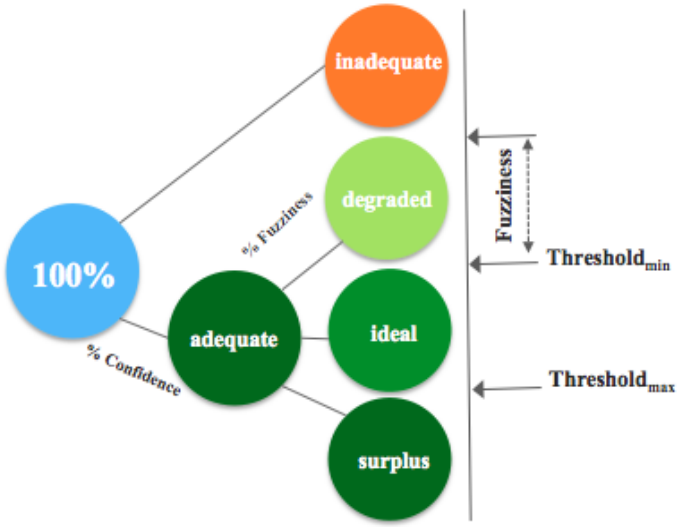


Figure 5.2: SLO evaluation in CSLA

Finally, we add an intermediate step in the evaluation process. This step consists to update the intervals classes using the nullification of degraded intervals by surplus intervals i.e., for each surplus interval we translate a degraded interval to ideal interval.

Listing 5.1: CSLA example.

```
1  <csla:terms>
2  <csla:objective id="GreenResourceSLO" actor="IaasProvider">
3  <csla:expression metric="Gr" comparator="in" threshold-min="25" threshold-max="30" unit="\%" monitoring="
       Mon-1" Confidence="91,66" fuzziness-value="5" fuzziness-percentage="18,18"/>
4  </csla:objective>
5  </csla:terms>
6  <csla:penalties>
7  <csla:Penalty id="p-Gr" objective="GreenResourceSLO" condition="violation" obligation="provider">
8  ...
9  </csla:Penalty>
10 </csla:penalties>
```

For the simplicity, we provide an example at Listing 5.1. Here we focus on only one SLO about the percentage of green resources (lines 1-4). The SLO states that the percentage of green resources should be guaranteed between 25 and 30, with confidence of 91.66%, fuzziness of 5% and fuzziness percentage of 18, 18%. In concrete terms, it means that the percentage of green resource measured within an observation period may be i) lower than 20% in 8.34% (100% - confidence% = 8.34%) of the observation periods, ii) between 20% and 25% in 16.66% (91.66% of 18.18%) of the observation periods and iii) greater or equal to 25% in 75%. A violation of the *GreenResourceSLO* implies a penalty that depends on the green percentage not respected (lines 7-9).

Using this objective for an evaluation window of 24 intervals, we accept 22 adequate

intervals (18 ideal and surplus, 4 degraded) and 2 inadequate interval.

5.3 Real-time Green Energy Management

In this section, we describe the model parameters and investigate the goal for cost reduction of spot green market as well as total energy expenditure by proposed algorithmic solution based on *Green SLA*.

5.3.1 Supply side characteristics

We consider our system operates in discrete time model. From day ahead REC market, IaaS provider purchase green energy for next suitable time period e.g., 12 hours or 24 hours. Furthermore, for evaluation and validation of SLA by CSLA language we divide before mentioned time frame in t (t = 12 or 24 hours time period). Moreover, the total time is divided into $J(J \in N^{+})$ coarse-grained time slots of each length of T, accordance with the length of the day-ahead REC market, e.g., minutes, hours in Figure 5.3. In addition, each fine-grained time slots τ, (τ = 30 minutes) are treated as monitoring window and t where, $t = jT(j = 1, 2, ..., J)$, can be defined as evaluation window for SLA validation in our model. IaaS provider purchase green energy from single or multiple REC providers (discussed in the previous chapter) in day ahead REC market for next 12 to 24 hours. So, for each fine-grained time slot, we define $d(\tau)$ where,$\tau \in (t, t + T - 1)$ is purchased with P_{max} upper bound price. As we integrate on-site renewable power source in our model, we consider $r(\tau)$ amount of green energy is produced and added to each fine-grained time slot. As green energy sources are very intermittent in nature, we suggest $r(\tau)$ to be $r(\tau) \geq 0$ for each fine-grained slot meaning, green energy will not be available in some slots due to the sporadic nature of the source. Hence, the supply side consists of two independent parts, *i.e.*, $U(\tau) = d(\tau) + r(\tau)$. As we consider, two time scale green energy market, if the demand of green energy is greater than the supply, IaaS provider has to buy additional energy in real-time from REC market which tends to have higher price on average than day-ahead or long-term ahead market similar to real-time electricity market. This additional energy is regarded as spot energy. We define $s(\tau)$ amount of green energy needs to be purchased from real-time spot green energy REC market at price $\beta(\tau)(O < \beta(\tau) \leq \beta_{max})$ in each fine-grained time slot if required.

5.3.2 Virtual energy model

At each fine-grained time slot, workload arrives with the requirement of green energy percentage e.g., 30%, that needs to be served and we define the request process as $e(\tau)$. We assume, non green energy can be drawn anytime from Grid if there is deficit of green energy in spot green energy market. Considering the demand and supply side, the ideal condition would be meeting exact demand from the supply side : $U(\tau) + s(\tau) = e(\tau)$ or

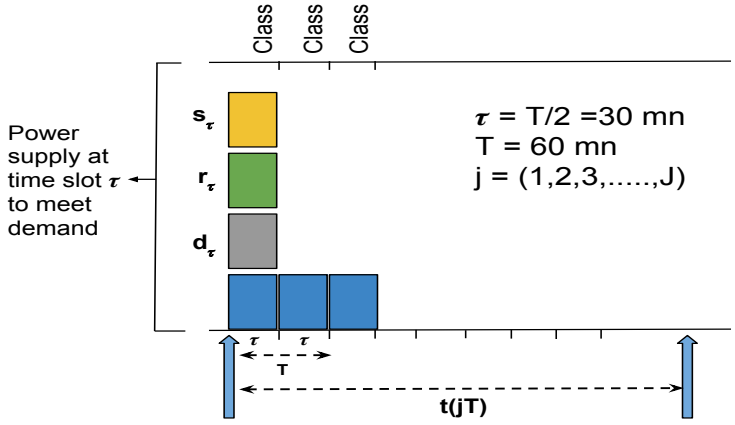


Figure 5.3: Supply side characteristics

$U(\tau) + s(\tau) \geq e(\tau)$ suggesting, supply might exceed the actual green energy demand in some slots. In our model, this superfluous energy will neither be stored in a storage nor be wasted, rather will be used as virtual energy in the data center. This additional energy will increase the percentage of green energy to the total energy. Hence, we characterize $b(\tau)$ as the summation of all available green energy present in the slot, which we can write as $b(\tau) = U(\tau) + s(\tau) = d(\tau) + r(\tau) + s(\tau)$. and define superfluous or virtual energy $v(\tau)$ as:

$$v(\tau) = \begin{cases} v & \text{if } b(\tau) \geq threshold_{max} \\ 0 & otherwise \end{cases} \tag{5.1}$$

5.3.3 Cost Minimization Problem of Spot Green Energy

As described in previous sub section, we consider, green energy demand, available green energy purchased from day-ahead green energy market and on-site plant, market price of spot green energy from a vector $(e(\tau), U(\tau), \beta(\tau))$ and $e(\tau)$,$\beta(\tau)$ is *i.i.d* over slots with some unknown probability distribution. Furthermore, $U(\tau)$, $e(\tau)$ and $\beta(\tau)$ are deterministically bounded by finite constraints U_{max}, e_{max} and β_{max}, so that: $0 \leq U(\tau) < U_{max}, 0 \leq e(\tau) \leq e_{max}$, and $0 < \beta(\tau) \leq \beta_{max}, \forall \tau$

Now letting $Q(\tau)$ represent the total green energy request in the queue on slot t, we will have following update equation,

$$Q(\tau + 1) = max[Q(\tau) - U(\tau) - s(\tau), 0] + e(\tau) \tag{5.2}$$

Here $s(\tau)$ is a decision variable (Amount of energy needed to buy from real-time spot energy market), which chosen in every slot τ to stabilize the $Q(\tau)$ depending on the current

state of the queue and vector$(e(\tau), U(\tau), \beta(\tau))$. We define a upper bound s_{max} for $s(\tau)$ as $0 \leq s(\tau) \leq s_{max}$. Hence, our objective is to design a flexible and robust control policy for time varying systems to formalize the stochastic cost optimization problem for spot green energy is mentioned below:

$$\text{minimize} \quad Cost_{av} = \lim_{t\to\infty} \frac{1}{\tau} \sum_{\tau=0}^{t-1} \mathbb{E}\{\beta(\tau)s(\tau)\} \tag{5.3}$$

$$\text{subject to} \quad \overline{Q} < \infty, \tag{5.4}$$

$$0 \leq s(\tau) \leq s_{max}, \quad \forall\tau \tag{5.5}$$

where, $\overline{Q}$ is the time average expected queue backlog, defined as:

$$\overline{Q} = \limsup_{t\to\infty} \frac{1}{\tau} \sum_{\tau=0}^{t-1} \mathbb{E}\{Q(\tau)\}$$

Since the virtual energy can not be present in every slot, the current control decision is coupled with the future decisions. As example, the deficit of green energy in some slots may be larger and hence IaaS provider has to pay penalty to SaaS provider or to the end client. We preferred Lyapunov optimization over dynamic programming to solve this optimization problem since dynamic programming requires significant statistics of demand and supply probabilities [FYH+15]. Furthermore, Lyapunov framework has been proven to be efficient to design control algorithms for aforementioned constrained optimization problem without requiring a priori knowledge of demand and cost statistics.

We propose two threshold parameter thr_{min} and thr_{max}, where $thr_{min} < e(\tau)$ and $e(\tau) < thr_{max}$. For instance, in our framework, thr_{min} and thr_{max} are 25% and 30% respectively. Since, energy demand can not be measured with actual number, we consider energy demand in every τ slot $e(\tau)$ to be in range of 5%, which is a fuzziness value that can be negotiated in the SLA phase (discussed at Section 5.2.2). Furthermore, the idea behind introducing the threshold parameters is to analyze whether any of the two events have occurred or not in the slot. When the supply side has lesser amount of energy than thr_{min} value to meet green energy demand in a slot, the slot is considered as a energy deficit slot. Hence, we characterize $b(\tau)$ such a way that, it can be either in between the value of thr_{min} and $thr_{min} - fuzzinessValue$ or lower than $thr_{min} - fuzzinessValue$ or above the value of thr_{min} and thr_{max} . Furthermore, in case of energy inadequacy aware virtual queue, the maximum allowable slots when the value of $b(\tau)$ lies on $b(\tau) < (thr_{min} - fuzzinessValue)$, can be defined as N_{max}. Therefore, the functionality of $b(\tau)$ will depend on thr_{min}, thr_{max} and *fuzziness value* which is constructed as:

$$f(b(\tau)) = \begin{cases} surplus(v(\tau)) & \text{when} \quad b(\tau) \geq thr_{max} \\ ideal & \text{when} \quad thr_{min} \leq b(\tau) \leq thr_{max} \\ degraded & \text{if } (thr_{min} - fuzziness \quad Value) \leq b(\tau) \leq thr_{min} \\ inadequate & otherwise \end{cases} \tag{5.6}$$

So, the functionality of $b(\tau)$ indicates that when there is excessive green energy available in a slot than demanded energy, we characterize those superfluous energy slot as surplus energy slot. We define a energy degraded aware virtual queue $X(\tau)$ and energy inadequacy aware virtual queue $Y(\tau)$ to measure the backlog of energy deficits in the queue by tracking the number of slots when energy deficiency and virtual energy is present. Moreover, for $X(\tau)$, having energy deficiency and presence of virtual energy can not occur simultaneously in a slot. For $Y(\tau)$, maximum allowable slots that can have energy deficiency in evaluation time window t is defined as N_{max}. Furthermore, the update equation for energy degradation and energy adequacy aware virtual queue will be:

$$X(\tau + 1) = max[X(\tau) - \gamma v(\tau), 0] + \gamma b(\tau) \tag{5.7}$$

$$Y(\tau + 1) = max[Y(\tau) - N_{max}, 0] + \gamma b(\tau) \tag{5.8}$$

where, γ is a counter, which adds values to corresponding parameter e.g., $v(\tau), b(\tau)$ whenever it is present in the queue. So, γ can be represent as $\gamma\{0, 1\}$. That means, value of γ is either 0 or 1. In other sense, when an interval is degraded, queue $X(\tau)$ will update 1 degraded interval, hence $\gamma b(\tau + 1) = [\gamma b(\tau) + (b(\tau) > 0)]$. We also update $\gamma v(\tau + 1) = [\gamma v(\tau) + (v(\tau) > 0)]$ to track of how many surplus interval (meaning $v(\tau + 1)$) is required to nullify degraded intervals. Similarly, when an interval is inadequate, queue $Y(\tau)$ will update 1 inadequate interval by setting $\gamma b(\tau + 1) = [\gamma b(\tau) + (b(\tau) > 0)]$. Therefore, this explains broadly the construction of equation (5.7) and equation (5.8).

5.3.4 Lyapunov Optimization

We define, $\Theta(\tau) = [Q(\tau), X(\tau), Y(\tau)]$ as the concatenated vector of actual and virtual queues. Moreover, the quadratic Lyapunov function is $L(\Theta(\tau)) = \frac{1}{2}[Q^2(\tau) + X^2(\tau) + Y^2(\tau)]$. So, the t slot conditional Lyapunov drift is interpreted as:

$$\Delta(\Theta(t)) = \mathbb{E}[L(\Theta(\tau + t)) - L(\Theta(\tau))|\Theta(t)] \tag{5.9}$$

Following the Lyapunov framework of drift-plus-penalty algorithm [GJNT06], our algorithm designed to observe the current queue states $Q(\tau)$, $X(\tau)$, $Y(\tau)$ and the current vector $(e(\tau), U(\tau), \beta(\tau))$ and to make a decision on $s(\tau)$ where $0 \leq s(\tau) \leq s_{max}$, to minimize an upper bound on the following expression in every τ slots:

$$\Delta(\Theta(\tau)) + V\mathbb{E}\{\sum_{\tau=t}^{t+T-1} \beta(\tau)s(\tau)\}$$

where, V is defined as a control variable as $V > 0$ which is chosen accordingly to IaaS providers goal to give different weights that affect operational cost and energy deficiency trade-off. A large deficit of energy can reduce the operational cost, but can have negative effects on green energy requirement in data center resulting high percentage of SLA violation contracted between IaaS and SaaS provider. So, our approach consider to minimize a weighted sum of drift and penalty.

Theorem 1 (drift-plus-penalty bound) *Let $V > 0$, $T \geq 1$ and $t = jT$, $\tau \in [t, t+T-1]$. For any control policy that satisfies $0 \leq s(\tau) \leq s_{max}$ for all τ and the demand backlog for t slots are $Q(t) < Q_{max}$, the drift-plus-penalty satisfies:*

$$\begin{aligned} &\Delta(\Theta(t)) + V\mathbb{E}\{\sum_{\tau=t}^{t+T-1}\beta(\tau)s(\tau)|\Theta(t)\} \\ &\leq BT + V\mathbb{E}\{\sum_{\tau=t}^{t+T-1}\beta(\tau)s(\tau)|\Theta(t)\} \\ &+\mathbb{E}\{\sum_{\tau=t}^{t+T-1}Q(\tau)[e(\tau) - U(\tau) - s(\tau)]|\Theta(t)\} \\ &+\mathbb{E}\{\sum_{\tau=t}^{t+T-1}X(\tau)[\gamma b(\tau) - \gamma v(\tau)]|\Theta(t)\} \\ &+\mathbb{E}\{\sum_{\tau=t}^{t+T-1}Y(\tau)[\gamma b(\tau) - N_{max}]|\Theta(t)\} \end{aligned} \tag{5.10}$$

where, B is a finite constant and compute the bound on above drift-plus-penalty expression which is defined as:

$$B = \frac{1}{2}max[(b_{max} - V_{max})^2 + (b_{max} - N_{max})^2 + (U_{max} - s_{max})^2 + e_{max}^2]$$

Proof: From the $X(t)$ update rule (5.7) we have,

$$X(t+1) \leq max[X(t) - \gamma v(t), 0] + \gamma b(t)$$

hence,

$$X(t+1)^2 \leq (X(t) - \gamma V(t) + \gamma b(t))^2$$

Therefore,

$$\frac{X(t+1)^2 - X(t)^2}{2} \leq X(t)(\gamma b(t) - \gamma V(t)) + \frac{1}{2}[\gamma b(t) - \gamma V(t)]^2 \leq \frac{1}{2}max[(b_{max} - V_{max})^2] + X(t)(\gamma b(t) - \gamma V(t))$$

Similarly by squaring equation (5.2) and using inequality,

$$(max[Q - \alpha, 0] + e)^2 \leq Q^2 + \alpha^2 + e^2 + 2Q(e - \alpha)$$

which holds for any $Q \geq 0$, $\alpha \geq 0$, $e \geq 0$, we get:

$$\frac{Q(t+1)^2-Q(t)^2}{2} \leq Q(t)(e(t)-U(t)-s(t)) + \frac{1}{2}[(U_{max}-s_{max})^2 + e_{max}^2]$$

Likewise, by squaring equation (5.8) and using inequality,

$$\frac{Y(t+1)^2-Y(t)^2}{2} \leq Y(t)(\gamma b(t) - N_{max}) + \frac{1}{2}[(b_{max}-N_{max})]^2$$

Combining above yields, we get t-slot conditional Lyapunov drift $\Delta(\Theta(t))$ as,

$$\Delta(\Theta(t)) < B + Q(t)[e(t)-U(t)-s(t)] + X(t)[\gamma b(t) - \gamma v(t)] + Y(t)[\gamma b(t) - N_{max}]$$

then, taking conditional expectation and summing the above inequality over $\tau \in [t, t+1+, ..., t+T-1]$, we obtain:

$$\begin{aligned}\Delta(\Theta(t)) < B + \mathbb{E}\{\sum_{\tau=t}^{t+T-1} Q(\tau)[e(\tau)-U(\tau)-s(\tau)] \\ +\mathbb{E}\{\sum_{\tau=t}^{t+T-1} X(\tau)[\gamma b(\tau) - \gamma v(\tau)] \\ +\mathbb{E}\{\sum_{\tau=t}^{t+T-1} Y(\tau)[\gamma b(\tau) - N_{max}]\end{aligned}$$

adding the operational spot green energy cost $V\mathbb{E}\{\sum_{\tau=t}^{t+T-1}\beta(\tau)s(\tau)|\Theta(t)\}$ on the both sides, given $\Theta(t)$, we prove the theorem.

5.3.5 Dynamic Algorithm

We minimize the right hand side of drift-plus-penalty at each fine-grained time slot $\tau \in [t, t+T-1]$ by observing queue statistics $Q(\tau)$, $X(\tau)$, $Y(\tau)$, green energy demand $e(\tau)$, green energy production $r(\tau)$, spot green energy price $\beta(\tau)$ and choosing $s(\tau)$ according to the following optimization:

$$\begin{aligned}\text{minimize} \quad & \sum_{\tau=t}^{t+T-1} s(\tau)[V\beta(\tau) - Q(\tau)] \\ & + \sum_{\tau=t}^{t+T-1} [(X(\tau)+Y(\tau))(\gamma b(\tau) - \gamma v(\tau))] \\ \text{subject to} \quad & 0 \leq s(\tau) \leq s_{\max}, \quad \forall \tau \\ & \sum_{\tau=t}^{t+T-1} \gamma b(\tau) \leq N_{\max}\end{aligned}$$

5.3.5.1 Algorithmic solution

The proposed Lyapunov framework runs Algorithm 1 in the background in a repetitive manner to ensure *Green SLA* according to the runtime context, namely the demand, the budget, the spot green energy market and SLA. The Algorithm 1 triggers the purchasing method. More importantly, it indicates the real situation to purchase energy from spot green energy market. It gives an edge over only choosing V parameter by procuring energy only when it is necessary to fulfill the contracted SLA.

Algorithme 1 : greenSla

Input : $interval.class, \beta(\tau), P_{spot}, Av_{spot}, N_{max}, ina_{curr}, deg_{curr}, idl_{curr}, sur_{curr}$
Output : $s(\tau), ina_{curr}, deg_{curr}, idl_{curr}, sur_{curr}$

```
1  s(τ) = 0;
2  if interval.class == inadequate then
3      if ina_curr < N_max then
4          ina_curr + +;
5      else
6          s(τ) = buy(β(τ), P_spot, Av_spot);
7          interval.class = updateClass(s(τ));
8          (ina_curr, deg_curr, idl_curr, sur_curr) = update(interval.class)
9  else if interval.class == degraded then
10     if sur_curr > 0 then
11         idl_curr+ = 2 ;
12         sur_curr − −;
13     else
14         if deg_curr < deg_max then
15             deg_curr + +;
16         else
17             s(τ) = buy(β(τ), P_spot, Av_spot);
18             interval.class = updateClass(s(τ));
19             (ina_curr, deg_curr, idl_curr, sur_curr) = update(interval.class)
20 else if interval.class == ideal then
21     if deg_curr > deg_max then
22         s(τ) = buy(β(τ), P_spot, Av_spot);
23         interval.class = updateClass(s(τ));
24         (ina_curr, deg_curr, idl_curr, sur_curr) = update(interval.class)
25     else
26         idl_curr + +;
27 else if interval.class == surplus then
28     sur_curr + +;
29 return s(τ), ina_curr, deg_curr, idl_curr, sur_curr
```

- *interval class==inadequate*: Line 2 of Algorithm 1 describes the *interval.class* for being at $((thr_{min} - b(t)) > fuzziness \quad value)$ and if $(\gamma b(t) < N_{max})$, we update the slot as

Algorithme 2 : buySpotMinCost

Input : $interval.class, interval.val, \beta(\tau), P_{spot}, Av_{spot}, penaltyModel$
Output : $class, cost$

```
1  if interval.class == inadequate then
2      if deg_curr < deg_max then
3          b = buySpot(Degraded, penaltyModel);
4      else if deg_curr > deg_max then
5          b = buySpot(Surplus, penaltyModel);
6      else
7          b = buySpot(Ideal, penaltyModel);
8  else if interval.class == degraded then
9      if deg_curr > deg_max then
10         b = buySpot(Surplus, penaltyModel);
11     else
12         b = buySpot(Ideal, penaltyModel);
13 else if interval.class == ideal then
14     b = buySpot(Surplus, penaltyModel);
15 class, cost=update(b);
16 return class, cost
```

Algorithme 3 : buySpot

Input : $class, penaltyModel$
Output : buy

```
1  buy = false;
2  if penaltyModel == static then
3      if demandedClass(class).isPossible() then
4          if cost(class) < Penalty(class) then
5              buy = true;
6  return buy
```

green energy inadequate slot. Otherwise, we purchase energy referring to line 6 by triggering Algorithm 2.

- *interval class==degraded*: Line 9 of Algorithm 1 indicates *interval.class* for being at $((thr_{min} - b(t)) \leq fuzziness\ \ value)$ and if any previous slot posses virtual energy $v(t)$, we nullify one degraded slot. Contrarily, line 17 triggers to Algorithm 2 for purchasing green energy if it is feasible.
- *interval class==ideal*: The *interval.class* in the line 20 indicates, if the value of $b(t)$ lies between thr_{min} and thr_{max}, no procurement is needed. But if previous degraded slots exceeded the threshold number (deg_{max}), we need to move to Algorithm 2 to purchase green energy. The number of degraded slots might be greater than the targeted deg_{max}, only if there is unavailability of green energy in the spot green energy market.

At the end, the algorithm updates the current intervals/slots status to either inadequate/degraded/ideal or surplus. We propose one method to purchase green energy from the spot green energy market named *buySpotMinCost* (see Algorithm 2). As the label suggests, the *buySpotMinCost* insists to minimize the cost by purchasing green energy to switch from one class to the next in the order (inadequate, degraded, ideal, surplus). In addition, the purchasing decision is based on the penalty model, cost and available quantity of energy in the spot green energy market. It can be observed from Algorithm 3 that, we support only static penalty model in this work. We buy only if the available green energy in the sport market is able to switch to the demanded class.

5.4 Evaluation

This section presents the results obtained from some experiments. In order to evaluate the proposed approach, we first describe our experimental environment. Then, we present cost analysis, SLA validation and how penalty model can influence the purchase decision and reduction of total expenditure in results section. Furthermore, insights and critical analysis are presented in discussion section.

5.4.1 Experimental Testbed

For a datacenter, Power usage effectiveness (PUE) is defined as the ratio of the data center's total power consumption to the data center's power consumption at the computer servers [GMR13]. Therefore, we consider a data center which has an average PUE of 1.77. Though some of the state-of-the-art techniques claim to have reduced this value closer to 1.20, still most of today's data center have higher PUE values than 1.7 [YM13]. Therefore, in the planning phase at Chapter 4 : Section 4.6.3, for transforming CPU utilization to power consumption, we traced CPU utilization for 7 days of 30 servers from PlanetLab [PP06] where CPU utilization has been traced for 500 different servers from across the world.

We have selected two server configurations with multi-core CPUs. The configuration and power consumption characteristics of the selected servers are shown in Table 5.1. So, we model *total data center power(t) = Server power consumption(t) × PUE value.* In addition, we use OpenForecast[2] to forecast power demand for next 24 hours based on last 7 days power consumption which was traced. As our goal is to make data center 30% green, we scale down the power requirement demand to 30% and dynamically buy the required green energy from day-ahead REC market from multiple providers. Furthermore, power requirements were transformed to energy requirement (power integrated over time), as energy is purchasable in Grid and REC market but not the power (at which rate, energy is transmitted).

Table 5.1: Power consumption by the selected servers at different load levels in Watt

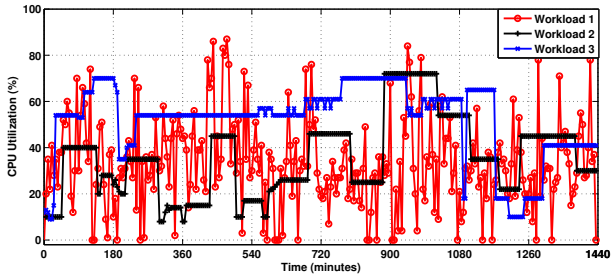
Servers	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Dell Inc PowerEdge M620	688	1151	1322	1494	1671	1848	2061	2289	2499	2765	3239
IBM NeXtScale nx360 M4	550	873	999	1123	1251	1380	1525	1673	1887	2116	2404

We have presented 3 kinds of workload in Figure 5.4(a). The first workload is a real workload traced from Planetlab for 30 servers, which can be seen as interactive jobs (e.g., airline booking, e-commerce site), the second workload is more characterized as an on/off pattern (e.g., scientific application or batch jobs for same modeled server). However, the third workload is created with greater forecasting error statistics compared to our predicted power workload in the planning phase by OpenForecast to evaluate how we can still propose a solution to fulfill green energy requirement based on *Green SLA*. The predicted green workload and above mentioned workload's characteristics is presented in Table 5.2. The first column represents the mean cpu utilization of the predicted and experimental workload. The third column indicates the average degree to which the data points differ from the mean. From the table 5.2 we can see that, third workload has higher variance indicating that the actual data points are quite spread in the data set while compared to predicted workload. Therefore, higher forecasting error exits in third experimental workload. Furthermore, we take advantage of the local solar irradiation data to calculate the amount of on-site green energy presented in the Figure 5.4(b). As the spot green energy market data is not available, we produce synthetic data to validate our experiment presented in Figure 5.4(c). There could be multiple energy consumers who might need green energy from the spot market, hence all the energy present in the spot green energy market will not be available for a single consumer, which makes a realistic assumption.

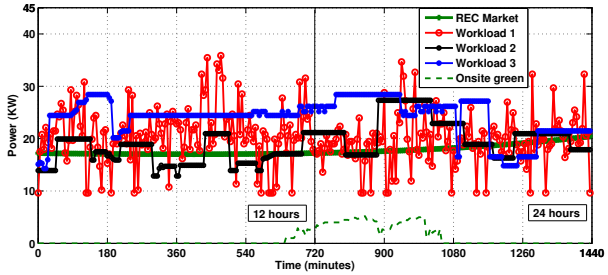
5.4.2 Cost function and algorithms for comparison

Finding market prices of each kWh produced by green sources are extremely difficult as most of the today's wind or solar power infrastructure or plants receive enormous

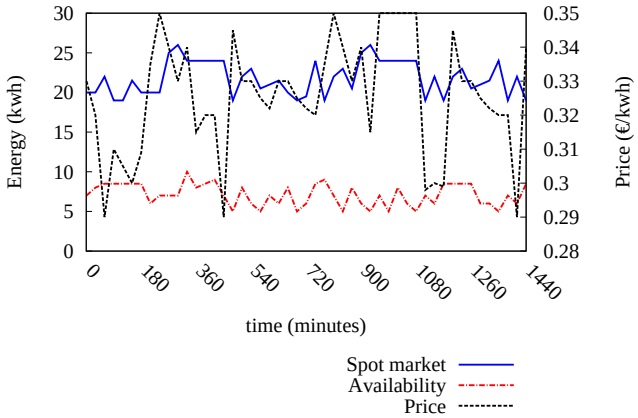
[2]http://www.stevengould.org/software/openforecast/index.shtml



(a) Workload



(b) Power Workload



(c) Spot market characteristics

Figure 5.4: Experimental Testbed

Table 5.2: Workload characteristics

Workload	Mean	Standard deviation	Variance
Predicted Green workload	17.75	0.93	0.86
Experimental workload 1	20.26	2.98	8.89
Experimental workload 2	19.15	3.47	12.05
Experimental workload 3	23.95	4.24	18.01

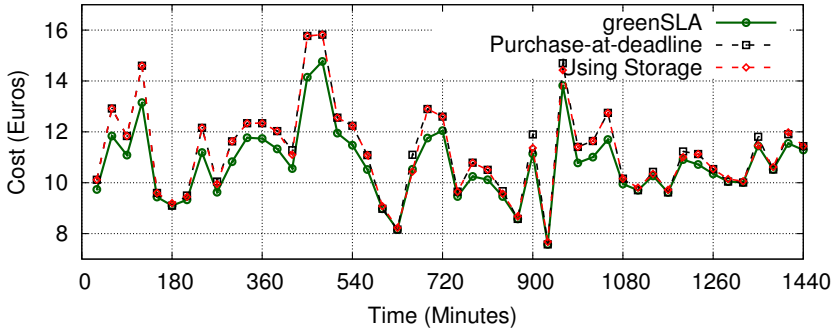
incentives either from government or different policy making organizations. Hence, to model a realistic price for energy in the day-ahead REC market, we investigate information of cap-ex, op-ex, levelized cost, fixed O&M cost, variable O&M cost of different sources of energy (e.g., Nuclear, Wind, Solar, Hydro etc)[3] and find that the ratio of energy consumption cost between nuclear/brown and green energy is *1:1.68* approximately. Since renewable sources are intermittent in nature, we consider the price of green energy sold at REC market will be in the range of *0.19 - 0.25* cents/kWh, which is *31.57%* in price variation, while the price of Nuclear or mixed energy provided by EDF[4] is 0.14 cents/kWh. As prices tend to be higher in the spot green energy market, we have made an assumption that green spot market price can be *30-35%* higher than the normal or day-ahead REC market. We compare our *greenSLA* algorithm with purchase at deadline approach and an energy storage approach that stores excessive on-site green and other abundant energy. Recent empirical studies shows that, the charging/discharging efficiency of a storage is $\eta = 80\%$ and cost per cycle is approximately 0.1 euro [UUJNS11]. In addition, we use fixed penalty value 1.5 euro/interval for IaaS provider if *Green SLA* is violated. We will analyze why do we choose this value and how it affects to the total energy cost in the result section. So, we define total energy cost as C_g,C_p,C_s respectively for *greenSLA*, "purchase at deadline" and "using energy storage" as follows:

1. C_g = grid energy cost + REC energy cost + spot green energy cost + penalty if violated SLA.

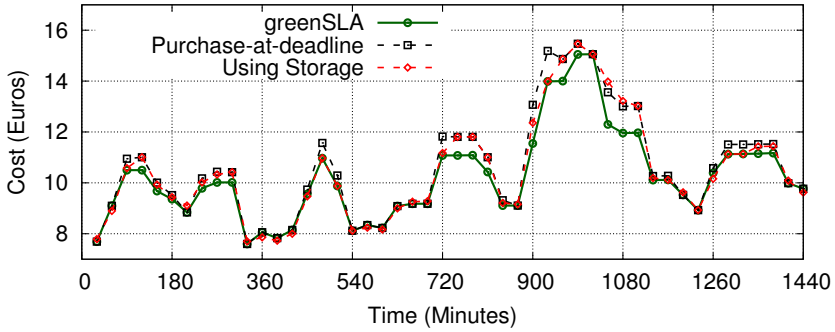
2. C_p = grid energy cost + REC energy cost + spot green energy cost

3. C_s = grid energy cost + REC energy cost + spot green energy cost + storage charging and discharging cost.

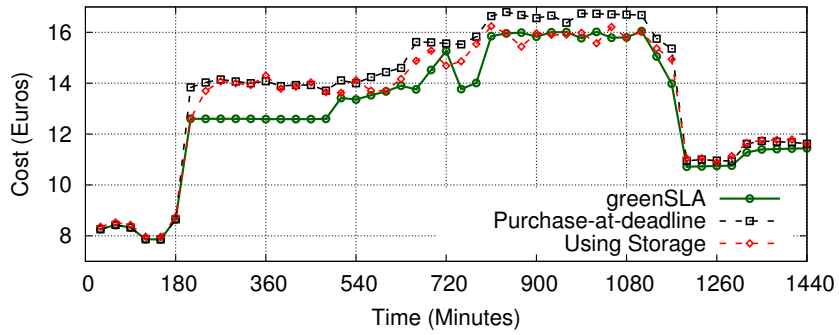
[3] http://www.eia.gov/forecasts/aeo/pdf/electricity_generation.pdf
[4] http://entreprises.edf.com/entreprises-45638.html



(a) Cost analysis for workload 1



(b) Cost analysis for workload 2



(c) Cost analysis for workload 3

Figure 5.5: Cost Analysis

Table 5.3: SLA between IaaS provider and its consumers

service	metric	oper.	thr_{min}(%)	thr_{max}(%)	fuzz.	% of fuzz.	conf.	penalty (euro/interval)
energy	green (Gr)	≥	25	30	5	91.66	18.18	1.5

5.5 Results

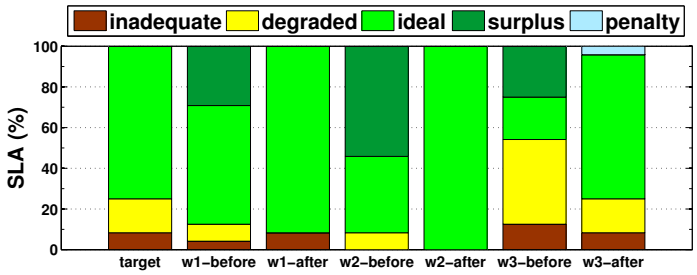
In this section we describe cost analysis, SLA evaluation based on our proposed *greenSLA* algorithm, impact of control parameter V (see section 5.3.4), impact of penalty to the total energy expenditure and robustness of our approach in detail.

5.5.1 Cost analysis

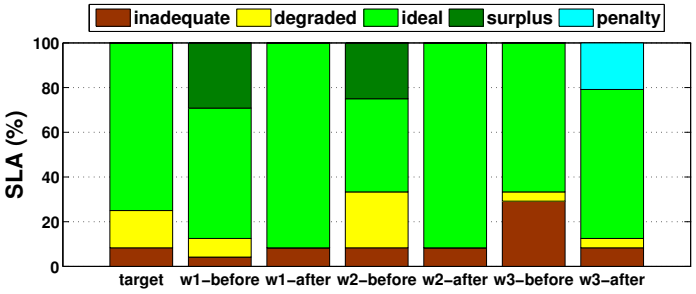
In Figure 5.5, we compare the cost for 24 hours between our proposed *greenSLA* algorithm and other two approaches namely "purchase at deadline" and "using energy storage". It is rationale that, purchasing green energy in every slot when it is needed incurs higher cost for purchase at deadline approach. Furthermore, the storage incurs additional costs due to charging the excessive energy and discharging the remaining energy in some slots. Besides, the storage does not discharge the exact amount of energy that has been charged earlier. From Figure 5.5(a) and 5.5(b), we calculate that, *greenSLA* achieves 4.47% and 4.17% cost reduction for workload 1 and 3.29% and 2.1% for workload 2 comparing to other two approaches. For workload 3, it has been noticed that (shown in Figure 5.5(c)), some time slots experience greater green energy inadequacy, hence *greenSLA* algorithm was forced to choose penalty for few slots. In some other slots, *greenSLA* chooses penalty over buying green energy from spot market, as the cost for buying green energy was slightly higher in terms of total expenditure. Since purchasing green energy option is limited in spot market for a single consumer, other two approaches cannot meet the exact demand. Nevertheless, *greenSLA* performs better by reducing 5.9% and 3.54% cost comparing to other approaches for workload 3. In terms of buying spot energy, Figure 5.6(c) shows, purchase at deadline approach incurs 8.17%, 6.26%, 15.62% expenditure of total energy cost for workload 1,2 and 3, the storage performs better by incurring 7.48%, 4.35% and 12.45% for respective workloads. In contrast, *greenSLA* significantly reduces the expenditure for spot green energy by only incurring .65%, .71% and 4.75% of total expenditure for above mentioned workloads. In our understanding, the concept of virtualizing the green energy leverages the process of reduction the total green energy expenditure by our algorithm than other two approaches.

5.5.2 SLA validation

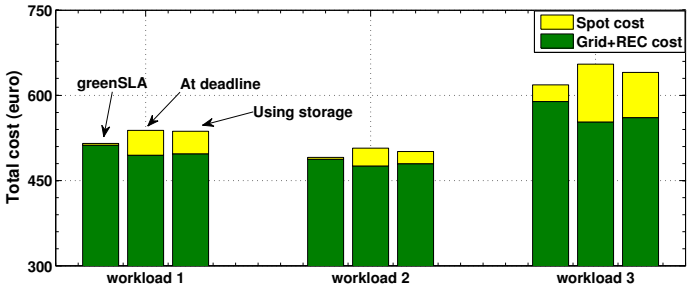
Figure 5.6(a) and 5.6(b) show the comparison of targeted and achieved SLA based on *greenSLA* algorithm, which is evaluated and validated through CSLA. For our experiment, we fix observation window $\tau = 30$ minutes and evaluate every 12 hours as evaluation window suggesting, we evaluate 24 intervals at a time. We present a example of CSLA at Section 5.2.2 having a SLA contract to provide 18 ideal intervals, 4 degraded intervals and 2 inadequate intervals out of 24 intervals or slots of green energy. The Table 5.3 summarizes the SLA between the IaaS provider and its consumers (SaaS providers). When



(a) SLA validation for first 12 hours



(b) SLA validation next 12 hours



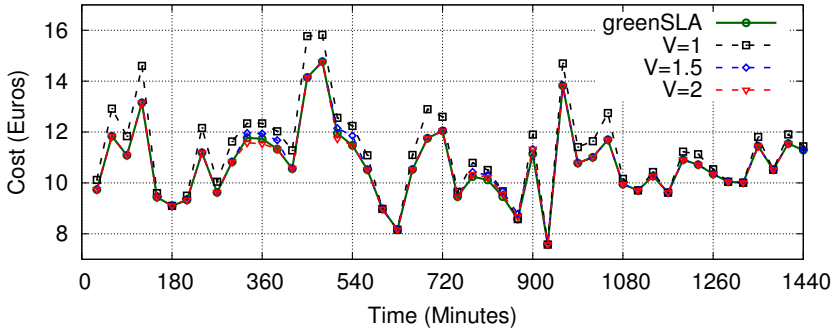
(c) Energy Cost

Figure 5.6: SLA Validation and Energy Cost

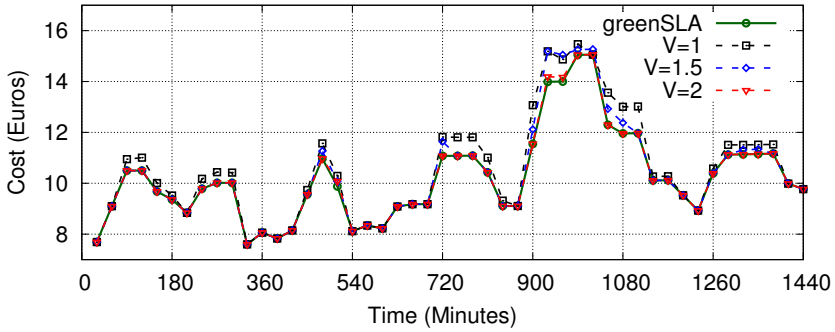
SaaS provider establishes a SLA with IaaS provider for having 30% green energy to run their workload, the Ideal intervals ranges from 25% to 30% for green energy in CSLA framework. We argue on the fact that, it is not possible to provide or measure exactly 30% green energy in each timeframe. Besides, running same workload over and over again in a server shows slightly different power consumption profile. Both the Figure 5.6(a) and 5.6(b) present the SLA target in the first column of the chart. Then we provide the actual interval status and evaluated results through CSLA. For example, *w1-before* represents the actual intervals without applying virtual green energy concept and *w1-after* indicates the evaluated interval results presented in percentage. For workload 1, *greenSLA* achieves exactly the target for first 12 hours, but shows better performance attaining 91.67% of ideal and 0% of degraded interval comparing to the goal of providing 75% and 16.66% of ideal and degraded interval respectively. The algorithm performs even better for batch jobs type workload 2, by providing 91.67% and 100% of ideal interval for first 12 hours and later 12 hours. Although the mean green energy demand for workload 1 and workload 2 deviates by 14.14% and 7.88% comparing to our predicted demand, *greenSLA* still managed to fulfill SLA by greater percentage, thus our algorithm is robust to inaccurate prediction information in terms of SLA validation. Due to the insufficient amount of green energy in the green spot market, *greenSLA* fails by 4.17% and 8.33% to meet SLA for workload 3 in two timeframe but still managed to incur lower cost than other two approaches, even though the algorithm choose to provide penalties in 6 intervals.

5.5.3 Impact of control parameter V

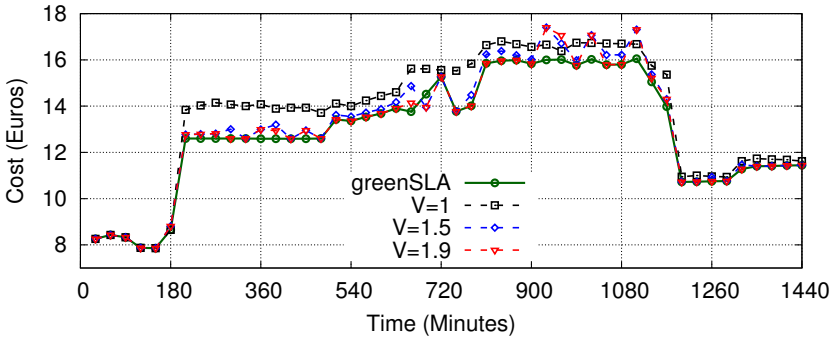
As shown in Figure 5.7, to simulate 3 kinds of workload, we fix t to be 24 hours and each fine grained timeslot as τ=30 minutes. We conduct experiments with different V values ranging from 1 to 5 and realized that, as the V value increases, it reduces the total energy cost. However, fixing larger value of V can violate contracted SLA. We see that, for workload 1, when $V = 4.2$, the control parameter performs well by reducing cost close to *greenSLA* and can satisfy the targeted SLA shown in 5.7(a). Nevertheless, if the value is increased by fraction, reduction of cost becomes larger but violates SLA. So this quantitatively indicates that, our proposed Lyapunov framework can approach very close to *greenSLA* within a diminishing gap of $O(1/V)$. Moreover we perceive that, the value of V can not be fixed ahead since it depends on the characteristics of the workload and SLA parameters. Figure 8.3(a) and 5.7(c) shows that, the same value of V can incur different level of costs. So, choosing the appropriate value is essential to make a trade-off between cost reduction and maintaining *Green SLA*. From our experiment, we find that the value of $V = 2$ and $V = 1.9$ can incur costs near to *greenSLA* for workload 2 and workload 3.



(a) V-values for Workload 1



(b) V-values for Workload 2

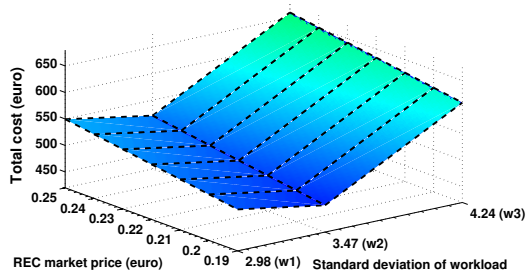


(c) V-values for Workload 3

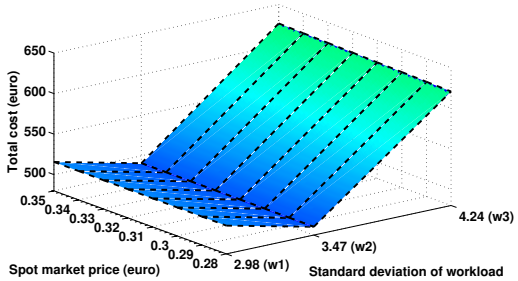
Figure 5.7: Impact of parameter V

5.5.4 Impact of penalty

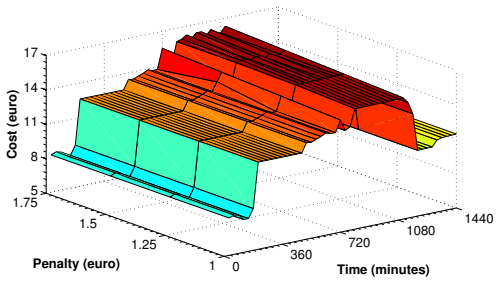
For our experiment, we choose static penalty per interval meaning, if IaaS provider fails to meet the demand of providing green energy beyond the SLA, the provider pay a fixed amount per interval. It is well understandable that, choosing a penalty value is extremely difficult and it depends on the business model of IaaS provider. Though we fixed the value of penalty as 1.5 euro/interval, Figure 5.8(c) shows that how the other penalty value affects the total expenditure for workload 3. In case of workload 1 and 2, *greenSLA* does not incur any penalty as it achieves the targeted SLA. We consider penalty value ranges from 1 euro/interval to 1.75 euro/interval by a factor of 0.25. We realize that, even increasing the penalty value by 0.25 euro/interval, the total expenditure increases only by 0.48%, 0.93%, 1.05% for 1.25, 1.50, 1.75 euro/interval. Hence, we choose 1.5 euro/interval penalty value which affects below 1% to the total green energy expenditure and certainly gives the flexibility to the IaaS provider.



(a) Total cost at various level of REC market price and energy demand variation



(b) Total cost at various level of spot green energy market price and energy demand variation



(c) Penalty vs cost over time

Figure 5.8: Impact of different energy prices and penalty to total cost

5.5.5 Robustness analysis

Due to the intermittent nature of green sources and price diversity in electricity market, we consider that, the REC and Spot green energy market energy price variation fluctuates in a range in Section 5.4.2. We observe that, with 31.57% (.19-.25 cents/kwh) price variation in REC market; the maximum cost difference fluctuates within [-4.13%, 6.63%] for 3 kinds of workload, whose mean and standard deviation (see Table 5.2) varies significantly than the predicted workload. Figure 5.8(a) shows the total cost curve in respect to different REC market energy price and energy demand variation. So, with the increase of energy price in REC market, the total expenditure increases slightly but expenditure can be increased significantly if the energy demand variation is large. The rationale is that, the cost reduction through greenSLA depends on the application workload. If the workload is more predictable, the cost reduction could be larger. Furthermore, Figure 5.8(b) indicates, the maximum cost difference fluctuates within [-1.43%, 1.30%] in respect to our proposed solution, while spot green energy market has 25% (.28-.35 cents/kwh) price variation. Section 5.5.2 shows that, *greenSLA* managed to fulfill SLA with greater percentage, even workload 1 and 2 has significant deviation of mean in terms of green energy demand. Therefore, *greenSLA* is robust and reliable to the energy prices in two time-scale market and energy demand, even though they have certain turbulence in variation.

5.5.6 Remarks

Furthermore, in this chapter, we only present one method to buy green energy from the green spot market based on proposed *Green SLA*. Our idea can be easily extendable to other methods required by IaaS provider to meet different goals and establishment for different SLA based on availability of green energy. Moreover, we propose only static penalty in case of SLA violation, but dynamic penalty can be integrated into the model as CSLA supports dynamic penalty modelling. Moreover, we do not propose an optimal solution as optimal solution for reducing cost of green energy can be varied depending on the workload pattern, on-site green energy generation and green spot market characteristics. From our experiments we observe that, it is possible to validate *Green SLA* with proposed spot green energy market characteristics. Nonetheless, providing penalty in most of the intervals/slots when green energy is not available seems a little unrealistic. In case of low availability of green energy, IaaS provider can apply energy efficient techniques to reduce energy consumption as well as green energy requirement in data center.

5.6 Discussion

This chapter proposes a brand new concept, namely *Virtualization of green energy* to tackle two state-of-the-art problems. They are : i) managing green energy in data center; ii) how to introduce Green SLA based on the presence and absence of green energy in data center.

Although in literature, the notion of "Green SLA" exists, none of the work are explicitly coupled with green energy. Furthermore, we extended SLA language namely CSLA to support the virtualization concept and to validate Green SLA. Likewise data center, any large consumer of power can adapt our proposed approach to make their infrastructure partially green.

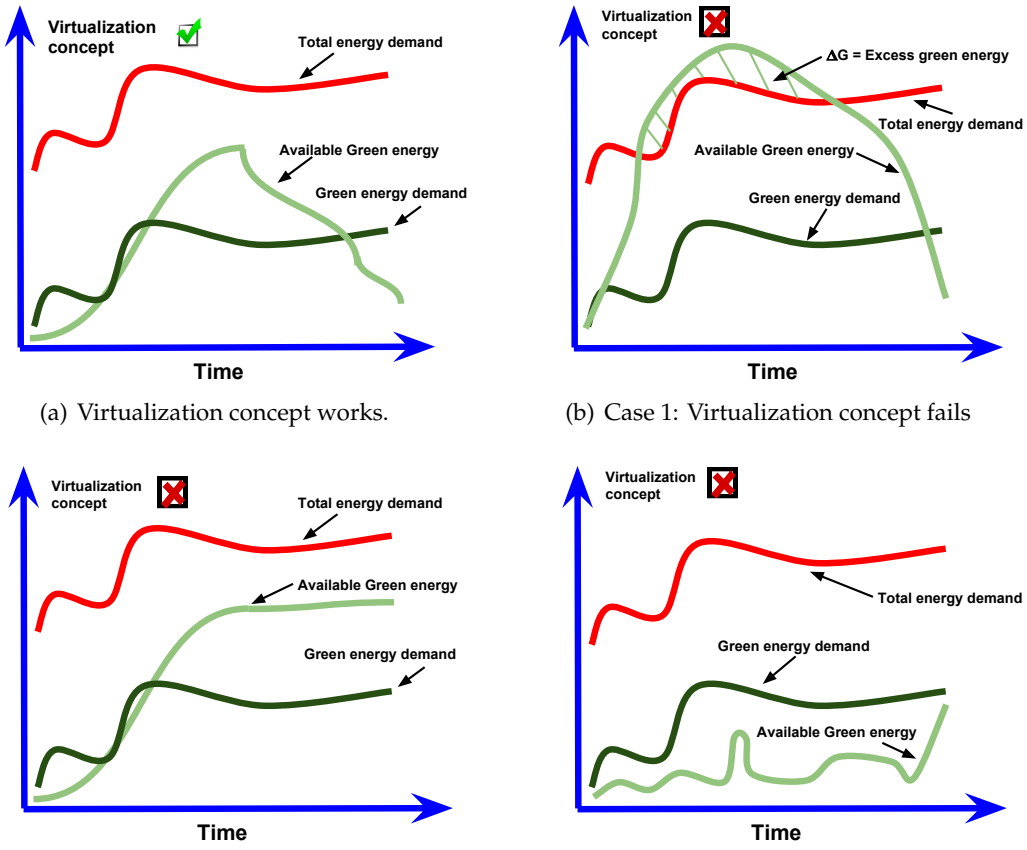
Limitation of Virtualization of green energy While the virtualization concept has been shown to be effective, it has certain limitations in three extreme cases. Those are as follows:

1. By revisiting the equation 5.1, we understand that, if there are superfluous green energy (*e.g.,* green energy production/availability is more than the green energy demand) in the data center, we consume the portion as virtual green energy. Even though, this concept overcomes the limitations of using energy storage or batteries, the amount of abundant green energy used in data center can be seen as stored in a *virtual battery*. Unlike any physical energy storage, virtual battery does not have energy losses for charging and discharging. But from theoretical point of view, alike any energy storage, virtual battery also has an upper bound of storing energy. If we characterize virtual battery as $v(\tau)$ in τ slot, then maximum energy that can be stored in virtual energy at τ slot will be $v_{max}(\tau) \leq (totalGreenEnergyAvailable - GreenEnergyDemand)$. But in case of excessive green energy production/availability that surpass the total energy demand, we can not use that portion of green energy at all. We characterize this excess energy as ΔG, indicated in Figure 5.9(b). Therefore, ΔG exists if $totalGreenEnergyAvailable > totalEnergyDemand$. In this kind of extreme case, our proposed Virtualization of Green energy concept fails.

2. Based on the definition of *Virtualization of Green energy*, our proposal is to smooth out the differences between deficit and surplus of green energy production/availability during a certain time window with the objective to obtain an summation narrowly superior to or exact to a certain threshold, which has shown at Figure 5.9(a). Hence, at the end of the evaluation of contracted *GreenSLA* via CSLA, our goal is to be borderline to the SLA. Our proposed concept helps to nullify the green energy degraded interval by the surplus green energy interval to achieve the contracted SLA goal. But, what happens if we find after evaluation that, number of surplus green energy intervals appeared more than the defined number of intervals in SLA? That suggests, lots of surplus intervals existed even after nullifying some degraded intervals. This is the indication of having more green energy than the green energy demand (*i.e.,* workload is very low than the predicted workload) for majority of the time (see Figure 5.9(c)). Therefore, the question arise that, what do we do with these surplus intervals?

3. In case of natural disaster, when possibility of having green energy from on-site, off-site or spot green energy market is very low, our proposed virtualization concept can not facilitate to abide the contracted SLA (see Figure 5.9(d)). If adequate and

surplus green energy interval does not exist, it is not possible to validate *GreenSLA*. So what would we do in this extreme case?



(a) Virtualization concept works.

(b) Case 1: Virtualization concept fails

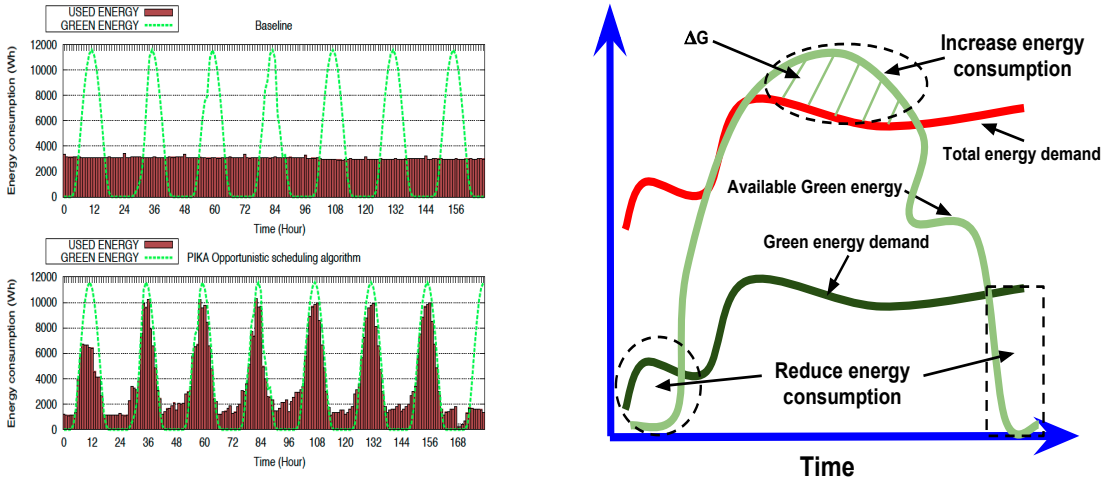
(c) Case 2: Virtualization concept fails

(d) Case 3: Virtualization concept fails

Figure 5.9: Limitation of Virtualization of green energy

Therefore, with the virtualization of green energy concept, which is a coarse-grained approach, three events can occur named as *"insufficient"*, *"ideal"*, *"overabundance"*. "Insufficient" indicates, lack of green energy in data center even though the virtualization was adapted. Whereas, "overabundance" event depicts of having more green energy in data center than which exceeds the virtual battery capacity. To succeed in reducing carbon footprint, all the cloud computing layers should participate in the process. Already, the state-of-the-art energy-efficient techniques at IaaS level are proposed and applied in academic and production environment. Therefore, very small room for improvement exists in IaaS level.

On the other hand, participating a SaaS application in energy saving or taking advantage of abundant energy, are missing from the literature. Traditionally, data centers host heterogeneous applications, such as batch and interactive application/jobs. Batch jobs arrive to the data center with deadlines, hence has the possibility to be scheduled to different times of the day by respecting the deadline when green energy is available. Thus, batch jobs have the capability to be paused and resumed by following green energy availability, which is shown in Figure 5.10(a) [LOM15]. On the contrary, interactive applications possess different constraints and challenges compared to batch jobs. Since interactive applications cannot be scheduled in advance, reconfiguration capability of interactive cloud application in run-time context can play a significant role. Hence, if we can trigger these events to interactive SaaS application and if the application inherits the capability to reconfigure itself based on the events, there can be an positive impact on energy consumption profile. Figure 5.10(b) expresses our goal to adapt an interactive application based on the aforementioned events. Therefore, our next chapter aims at providing insights and strategies needed to make interactive SaaS application *green energy aware* adaptive to consume less energy in case of insufficient green energy as well as to take advantage of abundant green energy for bettering quality of service or experience.



(a) Green energy adaptive batch application [LOM15]

(b) Proposal for Green energy adaptive interactive SaaS application

Figure 5.10: Green energy adaptive Cloud applications

Chapter 6

Creating green-energy adaptivity awareness in SaaS application

With the proliferation of Cloud computing, data centers have to urgently face energy consumption issues. Although recent efforts such as the integration of renewable energy to data centers or energy efficient techniques in (virtual) machines contribute to the reduction of carbon footprint, the smart usage of green energy in Cloud applications has not been yet addressed. By smart usage, we mean the awareness of a Software-as-a-Service application to increase energy consumption during the availability of green energy and to reduce energy consumption while green energy is scarce or absent. In this chapter, we propose a self adaptive auto-scaler architecture based on autonomic computing, which inherits the capability of sensing information as events from multiple layer while actions are performed only in application level. Thus, our approach can make an application adaptive by automatically adjusting to changing conditions. Furthermore, we investigate several application controllers based on different metrics (e.g., availability of green energy, response time, user experience level). Through extensive experiments and analysis with real application in real Cloud environment, smart usage of green energy is validated. We provide two hybrid controllers, that can provide formal guarantees of keeping the managed systems 95th percentile response time nearby the target, while brown energy consumption can be reduced as high as 13%. Moreover, our approach also adjusts the capacity requirement dynamically by releasing virtual resources to allow 29% more users to access the SaaS application. While these mentioned numbers can vary depending on workload and energy profile, it ensures the trend of the results.

6.1 Context and Motivation

According to *Nature climate change* [RMOR13], by the year of 2020, 15-30% decrease in carbon emission is required to keep the global temperature increase below 2 degree

Celsius. Therefore, aforementioned data are the indicator to build sustainable ecosystem around cloud services which involves different sub-systems of data center to heterogeneous hardware and software systems. Twofold ways to reduce carbon footprint of data centers at an acceptable level have been proposed in the literature. They are as following:

1. Firstly, explicit or implicit integration of renewable energy to the data center to increase the ratio and amount of green energy to the total energy to offset carbon footprint.

2. Secondly, a variety of research work have focused on environmental sustainability for Cloud Computing paradigm through energy consumption reduction by devising efficient strategies such as improving air cooling and humidification systems, using virtualization capability to increase server utilization and server consolidation [BAB12][HH13], workload migration [BJT+09], adopting DVFS [CWC14] [HASX07] etc. While all the work surface around data center power management and IaaS level, issues related to energy consumption in cloud applications have not been studied with requisite effort.

6.1.1 Why SaaS application should participate in energy reduction?

We argue that, a data center is only sustainable when each of the component of data center participates in the holistic process and push efforts for carbon emission reduction. Apart from that, there are several reasons to emphasis on energy reduction in SaaS layer are as follows:

Refactoring an application is difficult. One stream of work proposes conceptual reference models for building green and sustainable software applications that may lead to lower carbon emissions in the software life cycle stages [NDKJ11] [KDNH15] [LFF12]. While the proposed models are necessary for better sustainability requirements and practices, most of the modern cloud applications have complex codebase due to the tremendous rate of development activities. In 2013, Facebook reported that, the codebase for their front-end possess 10.5 million lines of actual code [FFB13]. Refactoring an existing cloud application based on these reference models is time-consuming and can have detrimental effects to the non-functional properties of a service. Therefore, the focus has shifted for readily designed, deployable and scalable cloud applications that can take leverage of underlying elastic infrastructures to prevent the service from saturation while unexpected event occurs *i.e.*, workload surge, hardware failures, etc.

Limitation of infrastructure elasticity. Resource elasticity comes with a greater cost and data center owner cannot augment infrastructure over night. Running more servers impact on energy density which results excessive heat that need to take out. With the increased

temperature in the data center, the reliability of servers and disks decreases, which in turn limits scalability [RSRK07]. Furthermore, cooling equipments have higher upfront and maintenance cost that can outstrip hardware costs when new servers and disks are added [Bar05]. Nonetheless, SaaS providers may find her/himself constrained with respect to the amount of resources to be used due to a predefined budget or design constraints. Thus, the question arises for cloud applications being sustainable enough to limit the need of adding more resources while existing infrastructure can handle the surged workload by respecting traditional Quality of Service (QoS) parameters. Preceding the cases, that leaves no choice but to dynamically adapt at service/application level during high intensity of workload or hardware failure to lower the needs of additional underlying resources, which can limit or cap energy consumption to a certain extent. Therefore, revealing and studying the impacts of cloud applications on energy consumption are required and finding solutions to reduce energy consumption are necessary for bettering the ecosystem's sustainability.

6.1.2 What makes energy reduction and adaptivity decision challenging?

With the increasing pressure on data center to achieve net zero carbon footprint or carbon neutrality, only reducing energy consumption is insufficient to reach the goal. Major tech companies like Google, Microsoft, Apple etc. are building data centers closer to renewable sources and buying carbon credits to offset carbon emission. Furthermore, research community prompted initiatives to optimize the usage of renewable energy [GKL+13] [LOM15] by scheduling batch type cloud applications *i.e.,* scientific workloads, delay tolerant map-reduce jobs etc. when abundance of green energy is present. On the contrary, most of the popular cloud applications are interactive applications which need instant responses *i.e.,* it should react with little to no latency, otherwise QoS can be seriously impacted. Since, interactive cloud applications can not be scheduled in advance, the only way to ensure the green energy awareness is to smartly adapt the applications internal with the presence and absence of the green energy.

However, there remains several challenges that antagonize the adaptivity decision to be strategized. Renewable sources are known to be very intermittent in nature, thus invoking an adaptation plan in advance can create discrepancy between the planned action and the run-time context of the application. Furthermore, from services point of view, response time and availability are the key metrics of interest for quantifying the performance and dependability of interactive cloud application. In contrast, making adaptation decision based on green energy availability without knowing the internals and current behavior of a application can lead to service saturation, hence providers can loss profit by preventing users from accessing the application. Therefore, formulating strategies in the presence of green energy while respecting traditional QoS parameters and pushing dynamic efforts to reduce energy consumption in the absence of green energy are complementary measures to improve the energy efficiency as well as to reduce carbon offsets.

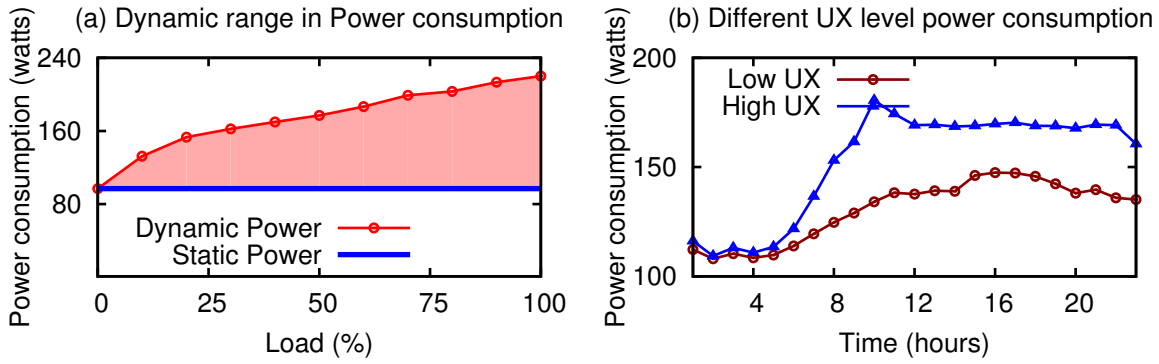


Figure 6.1: Power consumption analysis

6.2 How to make interactive SaaS application adaptive to green energy

Most of the popular cloud applications are well tailored to serve their clients with some extra features e.g., several product recommendation in an e-commerce application, which enhances user's Quality of Experience (QoE) but is not the core functionality of the service. For example, technology provider companies like amazon, ebay, spotify, netflix etc. provide several recommedations of similar or popular products to customers when they access the service. By identifying the independent but resource hungry software/application component which can be isolated to be activated/deactivated, could not only prevent transient behavior of an application in unpredictable runtime variations but also could reduce energy consumption when green energy is scarce.

To justify our statement, we followed two steps. First, we used stress benchmark to vary CPU utilization to measure the power consumption in Taurus cluster at the Lyon site of Grid'5000[1]. Figure 6.1(a) shows the static power consumption of an idle server and dynamic power consumption when load is increased. The difference of power consumption between an idle and a full loaded server is around 125 watts which implies an idle server consumes around 43.18% of total power while running at full load. Therefore, by manipulating resource hungry components in an application, the power savings could be minimal for shorter period of time. In contrast to the first step, we tested an extended version of RUBiS application in an aforementioned experimental setup providing low (no recommendation) and high user experience (multiple recommendations) and measured power consumption over time, as shown in Figure 6.1(b). The difference in energy consumption was on average 514.23 watts/hour after 24 hours, which corresponds to virtually turning off 6 idle servers

[1]A large-scale and versatile test bed for experiment-driven research. http://www.grid5000.fr

for an hour approximately. Therefore, in spite of the little amount energy that can be punctually saved while providing low user experience, satisfactory amount of energy can be saved for a long running service by applying dynamic adaptation in an application.

Therefore, by gradually increasing or decreasing user experience depending on the presence of green energy while attaining respectable performance is the only way to make interactive SaaS application adaptive to the presence of green energy.

6.3 Auto-scaler architecture

Autonomic computing has exhibited sheer promise for the ability to evolve a system to be self-adaptive by constantly sensing the system properties and tuning their performance and/or configurable parameters. We use the most popular self-adaptive design framework: Monitor-Analyze-Plan-Execute-Knowledge (MAPE-K) loop [KC03] for our auto-scaler. Our auto-scaler, continuously listens the instances of events *i.e.,* response time, green energy availability, working modes etc., pushed by SaaS and IaaS in a changing environment through different sensors, as shown at Figure 6.2 indicated by (1). Based on the received monitoring data of events which can be preprocessed or need post-processing, we decouple multiple events to extract pertinent information of the system behavior. Depending on the listened events pushed by the monitoring block, we analyze different events and if necessary, auto-scaler plans adaptation decision accordingly based on predefined configuration plan. We design several application controllers in the following sections which consist of *analyzing* and *planning* blocks (see Figure 6.2) of the MAPE-K autonomic loop by devising mostly reactive auto-scaling rules. Once the output of the configuration plan is ready, SaaS actuator executes actions to SaaS application via API calls in response to the deviation from the target system state *e.g.,* response time set point, consume less energy, quality of experience etc. which is showed at Figure 6.2 by (2). Additionally the knowledge block contains SLA parameters and the running state of the application.

Along with different performance (*e.g.,* QoS, QoE etc.) and resource aware metrics (*e.g.,* quality of energy), we propose three user experience levels. *Mode High* refers to high user experience while *Mode Medium* and *Mode Low* indicate to medium and low user experience respectively (see Figure 8.4(a)). When current application behavior deviates from target system state in terms of objective metrics, the auto-scaler gracefully downgrades the user experience from higher mode to lower mode and vice-versa through proper actuator value.

For example, usually popular e-commerce applications provide recommendation of the products to the users when they arrive or navigate to the site. In our case that is referred as *Mode 1* (see Figure 8.4(b)). During the low variability of workload or the presence of abundant green energy, we switch the application to higher mode, which we consider as *Mode 2*. In contrast, during the high variability of workload or scarcity of green energy availability or resource capacity, we switch the application to zero recommendation (*Mode 0*).

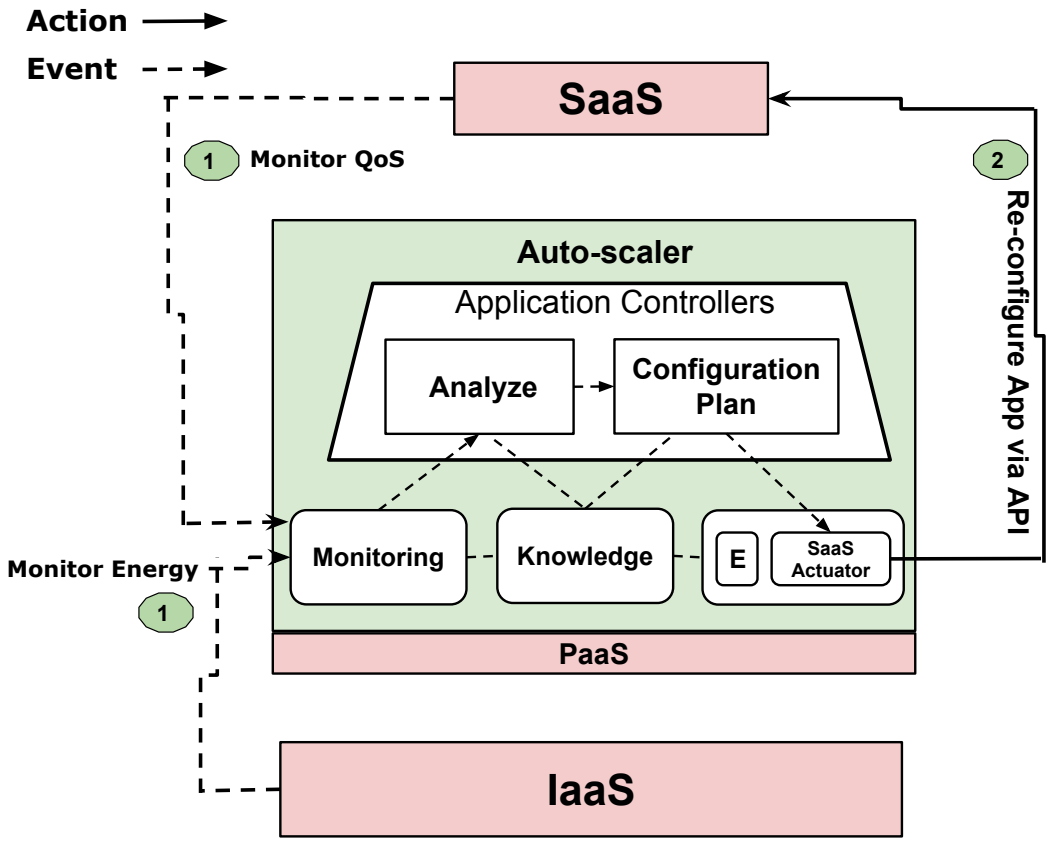
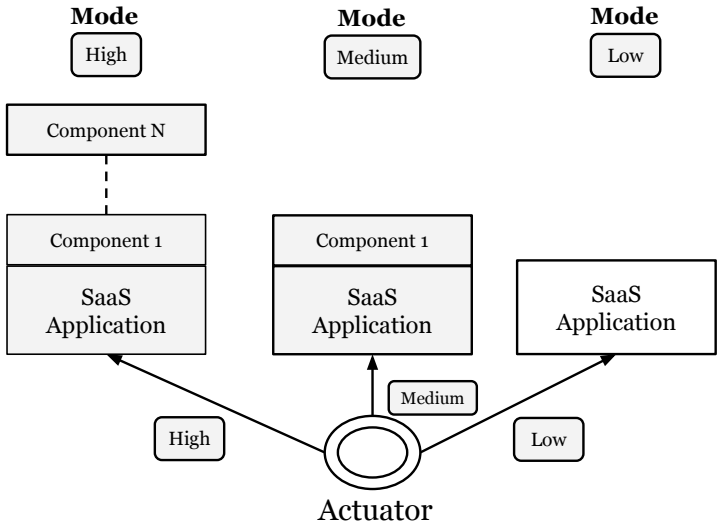


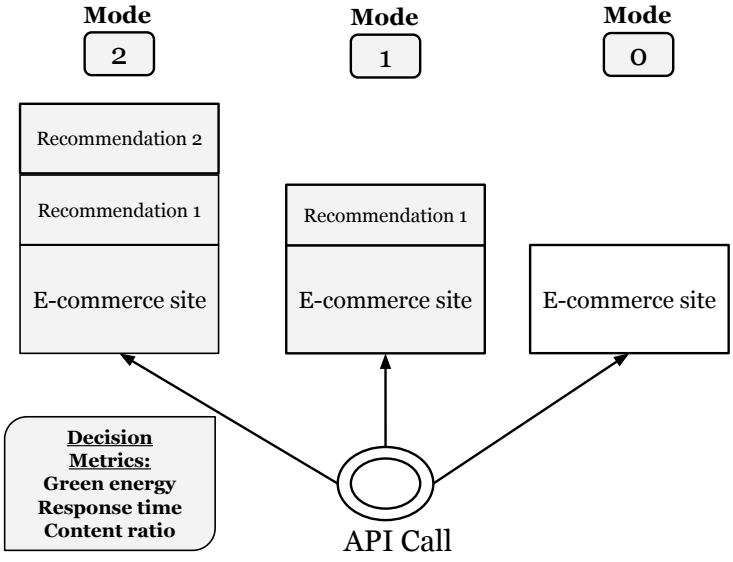
Figure 6.2: Auto-scaler architecture

6.4 Single metric application controllers

Static threshold based policies are one of the most popular model among the biggest Cloud providers like Amazon and third party tools such as RightScale for designing auto-scaling techniques. To design our auto-scaler, we consider Event-Condition-Action which is a rule based approach instead of robust control formulas since ECA rules are convenient to understand and can be implemented by administrators very easily. Furthermore, our auto-scaler is in charge of making decisions based on a single metric, and taking actions in SaaS layer, without human intervention. We discuss our controller's behavior in the presence of different goals in the following sub-sections.



(a) Example of different application modes



(b) Realistic example of application modes

Figure 6.3: Application modes under different service level

6.4.1 Green energy aware controller

We intend to build a controller which can make adaptive decision based on the availability and the quality of energy. Recalling from the Chapter 5 - Section 5.6, three events can occur while the concept of virtualization of green energy is adopted, namely, "insufficient", "ideal" and "overabundance". When *ideal* event is triggered from the infrastructure, the controller chooses an actuator value that prompts the medium user experience mode (mode 1), and in case of *insufficient* event, mode 0 is activated. In contrast, Mode 2 is activated when *overabundance* event is passed to the controller, which is showed in Algorithm 4. The idea behind is to consume more energy in the period of abundant green energy by following the green energy event.

Algorithme 4 : Green energy aware controller

Input : Thr_{max} = Threshold for green energy, *event* = <"insufficient","ideal", "overabundance">, $Curr_{GE}$ = Current green energy production
Output : $Curr_{mode}$ = Current application mode.

```
1 if (handleEvent == greenEnergy) then
2     if event == "insufficient" or Curr_GE == 0 then
3         app.mode ← mode 0
4     else if event == "overabundance" or Curr_GE > Thr_max then
5         app.mode ← mode 2
6     else
7         app.mode ← mode 1
8     Curr_mode = app.mode
9 return Curr_mode
```

Apart from virtualization of green energy concept, our controller can be adaptable if a provider has an on-site renewable power sources. On-site energy sources are only available during certain times. For instance, solar energy is available during the day and the amount produced depends on the weather and the season [GLN+12]. Due to the intermittency, we can divide the total green energy production to three different regions to treat as events as mentioned before, *i.e.*, no green energy at night (*insufficient*), few to adequate energy at early morning to the late afternoon (*ideal*) and substantial amount of energy at mid-day (*overabundance*). To distinguish between the regions we can choose a static threshold Thr_{max}, above which the controller activates high user experience mode (mode 2). When green energy production *e.g.*, $Curr_{GE}$ falls between 0 and Thr_{max}, the controller chooses an actuator value that triggers the medium user experience mode (mode 1), and in case of current green energy amount is null, mode 0 is activated.

Therefore, system must be adaptive to exploit the presence and absence of green energy to offer differentiated user experience to the end-clients. Since this approach does not consider any feedback from the internals of the application or make any adjustment due

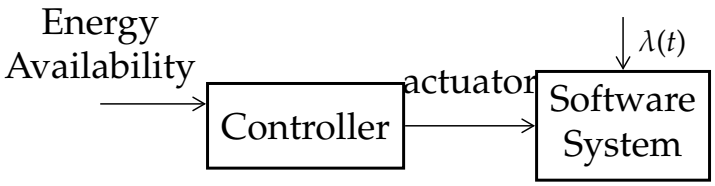


Figure 6.4: Green energy aware controller

to the disturbance to the system (*e.g.*, number of request that is denoted by $\lambda(t)$) while decision making, the controller works as an open loop controller (see Figure 6.4).

6.4.2 Response time controller

Response time is an essential metric to guarantee cloud based service performance. Our goal is to keep response time under certain threshold dynamically to maximize availability of the service in unpredictable and variable workload condition. Therefore, we closed the managed software system by a feedback loop, where in each control period, the output is forwarded as a Map of response time and workload arrival rate to compare with the target set-point, which is shown at Figure 6.5.

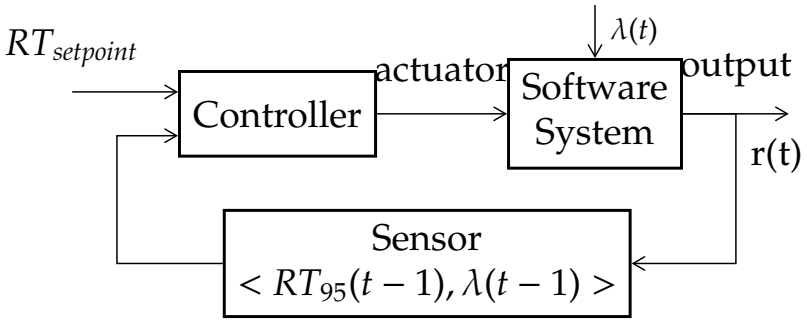


Figure 6.5: Response time aware controller

We measure the 95th percentile response time since it provides better approximation of entire distribution than the average response time. Afterwards, the information is forwarded to compute a function;

$$E_{rr}(t) = 1 - \tilde{\lambda}(t) * \tilde{r}(t) \tag{6.1}$$

Where $\tilde{\lambda}(t) = \frac{\lambda(t-1)}{\lambda_{median}}$; $\tilde{r}(t) = \frac{RT_{95}(t-1)}{RT_{setpoint}}$

In cloud environment, unpredictability and burstiness of user requests is common phenomena. Should it be not realized by the system or predicted in advance, can dramatically degrade application performance. Therefore, workload arrival rate acts as a disturbance, $\lambda(t)$ to the system. For capturing the change in the arrival rate, current arrival rate in the system is divided by median of previous arrival rates. A median filter is used with window size of four, that provides better estimation about variability of the workload arrival rate. $\tilde{\lambda}(t)$ provides the indication of acceleration and decelaration of user requests and when it is multiplied with $\tilde{r}(t)$, it helps to realize the increment or decrement of the systems's near future response time. Therefore, the idea is to keep $E_{rr}(t)$ function greater than *zero* to stabilize the system to operate under target response time. Therefore, in variable load condition, actuator's value must be controlled in a way to satisfy $\tilde{\lambda}(t) * \tilde{r}(t) < 1$. When few users accessing the application, $E_{rr}(t)$ function can be closer to *one*, hence we define a threshold value ($0 < Thr_{rt} < 1$) above which the system can provide high user experience, that is *mode 2* (see line 6 of Algorithm 5). Between *zero* and threshold value, the controller selects medium user experience mode. When the function becomes less than *zero*, all the recommendation is disabled to reduce the current response time to the targeted value, which corresponds to low user experience in the client side.

Algorithme 5 : Response time aware controller

Input : $Thr_{rt}, \lambda = [0 \quad 0 \quad 0 \quad 0], setPoint, app$
Output : updated λ, $Curr_{mode}$ = Current application mode.

```
1  if (handleEvent == responseTime) then
2      λ(t − 1) ← servedRequest
3      enqueue(λ)
4      function ← 1 − (λ(t − 1)/λ_median) ∗ (RT_95/setPoint)
5      if (function > 0) ∧ (function < Thr_rt) then
6          app.mode ← mode 1
7      else if function ≤ 0 then
8          app.mode ← mode 0
9      else
10         app.mode ← mode 2
11     dequeue(λ)
12     Curr_mode = app.mode
13 return λ, Curr_mode
```

6.4.3 QoE based controller

SaaS provider can have different quantifiable non-functional goals in web based interactive applications *i.e.*, quality of the contents, tracking user's navigational activity report or personalized recommendation, etc. These goals can be quantitatively expressed by SaaS provider by defining and implementing the possible strategy in an application

controller. Moreover, by increasing users experience, SaaS provider can generate higher revenue [FHB10]. From SaaS provider's viewpoint, it becomes critical requirement in modern services and can introduce new class of Service level objective (SLO). For example, an e-commerce service could have a goal - "Serve Recommendation" stating "SingleRecommendation(x%)" and "MultipleRecommendation(y% of x%)" of total requests in a SLA.

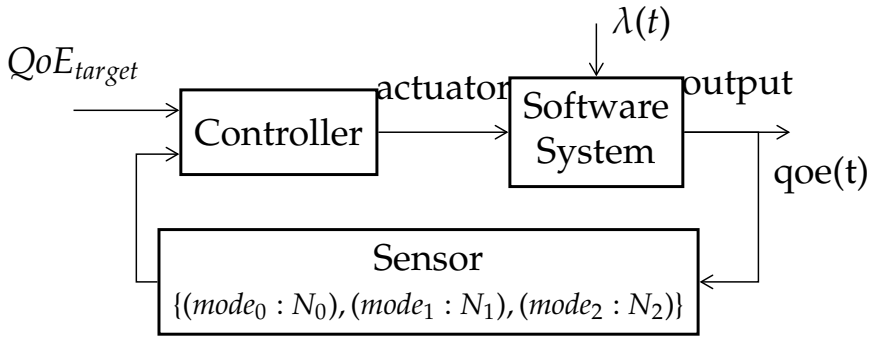


Figure 6.6: QoE aware controller

To satisfy the goal, we close the feedback loop with sensing the information as a map of how many requests were served with different modes in last control period, which is shown at Figure 6.6. Based on the information, the controller computes the current percentage of recommendations provided by the system and choose actuator value accordingly (see Algorithm 6). If the targeted percentage of recommendation is met, for example, "SingleRecommendation(x%)" is achieved, then actuator value is chosen accordingly to satisfy "MultipleRecommendation(y% of x%)". Theoretically, all the requests that were served in between two control period should have same mode *i.e.*, actuator value; for instance, if the actuator value was 1, then all the requests were entitled to mode 1. However, if the control time is small and two consecutive control period's actuator value is different, some requests which have arrived just before the ending of a control period can still wait in the queue to receive information from another tier of the application. That implies, few requests can have different associated modes in same control period.

6.5 Evaluation

In this section, we present the evaluation results of single metric controllers and their impact on cloud based application in terms of response time, quality of user experience and energy consumption. The goal is to advocate the benefits and limitations of each controller while experimenting with real cloud application and real workload traces.

Algorithme 6 : QoE aware controller

Input : Map M, Thr_1 = Target percentage for recommendation 1, Thr_2 = Target percentage for recommendation 2, $Dist_1 \leftarrow 0$ = Difference between target and current recommendation1 percentage, $Dist_2 \leftarrow 0$ = Difference between target and current recommendation2 percentage.

Output : $Curr_{mode}$ = Current application mode.

```
1  if (handleEvent == QoE) then
2      Total_Per1 = TotalReq_mode1/Total_req
3      Total_Per2 = TotalReqmode2/Total_req      // Number of requests in each mode is stored in a
       file in each iteration to update and compute Total_Per1 and Total_Per2
4      update(Dist_1, Dist_2)
5      if (Total_Per1 > Thr_1) ∧ (Total_Per2 < Thr_2) then
6          app.mode ← mode 2
7      else if (Total_Per1 < Thr_1) ∧ (Total_Per2 > Thr_2) then
8          app.mode ← mode 1
9      else if (Total_Per1 < Thr_1) ∧ (Total_Per2 < Thr_2) then
10         if Dist_1 < Dist_2 then
11             app.mode ← mode 2
12         else
13             app.mode ← mode 1
14     else
15         app.mode ← mode 0
16     Curr_mode = app.mode
17 return Curr_mode
```

6.5.1 Infrastructure configuration

The experiments were conducted in Grid'5000 Lyon site, with 2 physical machines linked by a 10 Gbit/s Ethernet switch and connected to wattmeter. Each machine has two 2.3GHz Xeon processors (6 cores per CPU) and 16GB of RAM, running Linux 2.6. Openstack Grizzly 1.0.0 was used as platform, which requires one dedicated physical machine for the cloud controller management system. Consequently, the second physical machine was used as compute node to host VMs, which in turn, are pre-configured to run Ubuntu 12.04.

6.5.2 Application configuration

In Brownout [KMAHR14], authors provided a user-to-user recommendation engine which can enhance user experience. Along with that, we implemented a fairly simple item-to-item recommendation, to offer better user experience, which is showed at Listing 6.1.

Listing 6.1: SQL statement for the recommender system.

```
1 SELECT
2     items1.id
3 FROM
```

```
4       items AS items1.id
5       JOIN comments AS c ON items1.id = c.item_id
6       JOIN items AS i2 ON items1.category = i2.category
7     WHERE
8       i2.id = :current_item_id AND
9       items1.nb_of_bids >= i2.item_id AND
10      items1.id != :current_item_id
11  ORDER BY rating DESC
12  LIMIT 10;
```

The simple recommendation engine provided at Listing 6.1 can be summarized as *"Retrieve 10 products from same seller and same product category which has higher or same user bid count with high customer rating"*. Although both the recommendation engines lack the sophistication and worldly complexities, they do serve as a reasonable example of providing user experience that a cloud application can isolate from core functionality of the service to activate or deactivate at runtime. Since aforementioned recommendation is added in the `ViewItem.php` page of RUBiS application, we focused on enabling or disabling these recommendation components by adding a function called *getMode()*. The function reads a file, where actuator value is updated in each control period and execute the associated modes for each user request. We also have added a html parser function in `PHPprinter.php` page to extract information of which request were served with which mode. For instance, Mode 1 activates the code of recommendation one, mode 2 activates both recommendations and mode 0 provides no recommendation. Furthermore, the extended RUBiS application was deployed in SaaS fashion and architecturally organized in 3 tiers: load balancer, web and database (db) tier. The compute node consists of one VM having Nginx load balancer, which distributes the request across three VM's of Nginx application server (see Figure 6.7), each having 1 cpu core and 2GB of memory and a single VM of MySQL db server of 8 cpu core and 16GB memory.

6.5.3 Auto-Scaler

Our auto-scaling solution is hosted inside the cloud controller machine. As shown at Figure 6.8, logstash agent is running at the Load balancer to collect separate field of metrics from access log (*i.e.*, response time, working modes of requests) and ships to a message broker (redis server) for data ingestion and storage at the controller node. Furthermore, indexer retrieves the data from the message broker, apply configured filtering to process 95th percentile response time, how many requests were served in last interval and associated mode of requests. Interval varies from 20 to 60 seconds in our experiments. In contrast, green energy information is pushed by the infrastructure through an API. The knowledge part contains the target response time and SLA parameters. Finally, based on all the aggregated data and decision metrics, the auto-scaler updates the appropriate actuator value via an API, which overwrites the file that resides in each VM of the web tier.

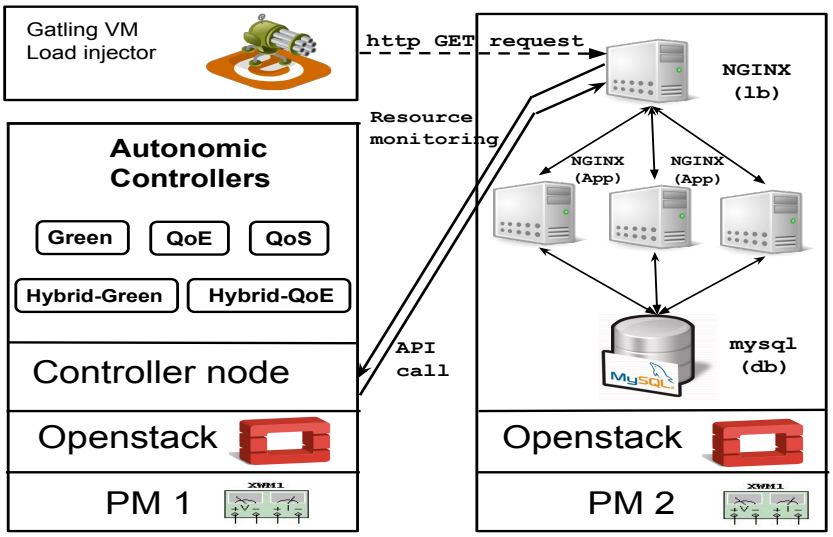


Figure 6.7: Experimental Testbed

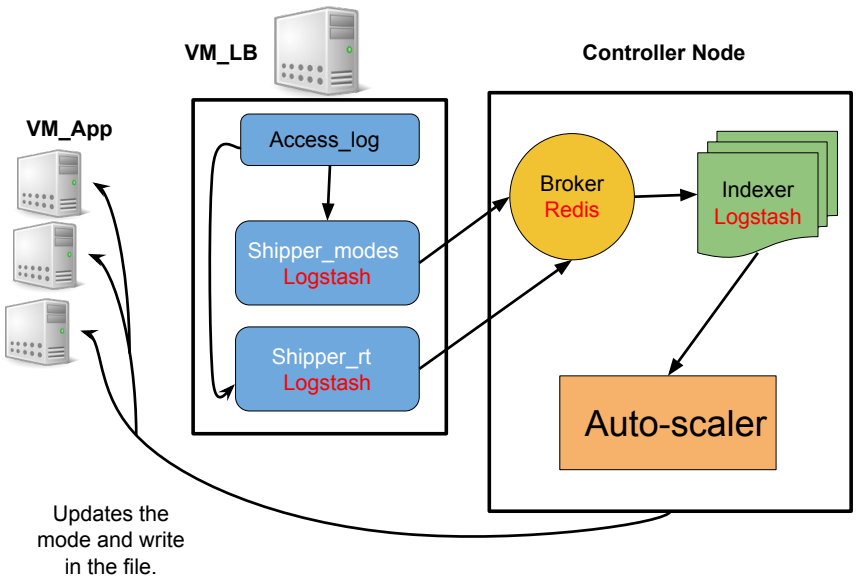


Figure 6.8: Monitoring

6.5.4 Workload traces

We took the real trace of wikipedia german page of one day and 1998 fifa world cup website traffic over one and half months [FJLB15] and scaled the data set to fit with our experiment, which is showed in Figure 6.9. While the wikipedia trace has a steady and incremental pattern of requests over time, fifa trace possesses some heavy temporal spikes. We relied on Gatling[2] as a load injector to generate our desired workload. To generate the workload, we choose an open system model, where user request is issued without waiting for other users response from the system. Furthermore, we emulated read-only workload where each user arrives to the homepage, browse any item category from a vast catalog, click on a product to extract its information, view seller rating and his/her reputation related to the product. We have kept request timeout to 32s. We traced the solar energy production that was added to the grid for one day (12th April,2016) from EDF, France[3] and scaled the values suited for our experiment. As discussed at Section 6.4.1, both the virtualization of green energy concept and on-site green energy can be treated as the same way if event-condition-action is adopted. Furthermore, the duration of each experiment was 96min and each was run several times. We considered 96min as 24 hours, *i.e.*, each 4min in our experiments correspond to 1 hour.

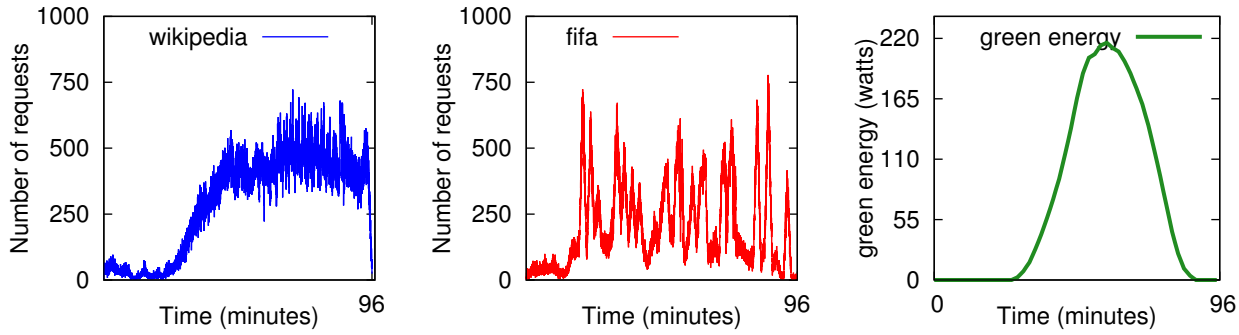


Figure 6.9: Workload trace

6.5.5 Results

In this section, we show and analyze through our experiments how single metric controllers can respect their goals and can outperform a non-adaptive approach. Non-adaptive approach always provides multiple recommendations irrespective to the workload changes, user request failure, SLA violation etc. Since, most of the popular cloud based applications lack reconfiguration capabilities at run-time, we realized this state of the application as non-adaptive.

[2] http://gatling.io/
[3] http://www.rte-france.com/fr/eco2mix/eco2mix

6.5.5.1 Response time

In Figure 6.10, we grouped response time by taking average over minutes. For wikipedia workload, as the workload increases around 40th minute, the response from the system grows big thus the system became unresponsive for non-adaptive approach. Even though the workload decreased at the very end of the experiment, the response time was never recovered to a steady state. The reason being that, the issued request would still run in the database server and waste the computing resources, although the transaction may timeout soon, or that may have already been timed out [KPD+14]. Figure 6.12(a) illustrates that, the 95th percentile response time of this approach is around 30.2 seconds[4]. Figure 6.10 shows, very high number of requests were failed. Out of 1.7 millions of requests which were injected, approximately 490k requests were failed, which accounts to 29% of requests and half of rest of the successful requests faced very high response time, *i.e.,* beyond 2 seconds. As a result, the service provider will not only lose money, but also unhappy customers can join to their competitors, incurring long-term revenue loss and lowering their reputation. Similarly, this approach increases the response time to 6 seconds on average while the temporal workload peaks appeared in fifa workload. Figure 6.12 shows the 95th percentile response time incurred by this approach is around 6.8 seconds, which is out of the plot.

Since the tolerable response time has been considered to 2 seconds [NH07], we set 1 second as target set point to allow a safety distance from being overloading the system and the threshold value to .5 for the *response time controller*[5]. We choose the intermediary value to be unbiased to provide single to multiple recommendations. We fixed control period as 20 seconds, which means that the controller collects 95th percentile response time periodically from the managed system and take actions if necessary. While Figure 6.10 shows that the controller keeps the average response time around 500 milliseconds (ms) in high number of requests period for both the workload profile, the 95th percentile response time is 750ms and 350ms for wikipedia and fifa workload respectively, as shown in Figure 6.12. Furthermore, several experiments were run and we found only 40-100 requests failed on average, which ensures 99.99% availability of service. Thus, availability of the service can be improved by 29% compared to non-adaptive approach.

On the contrary, the *Green controller* reaches to high response time region at 56th minute in for wikipedia workload (see Figure 6.10). We choose 60 seconds as control time in our experiment, since the green energy availability does not change abruptly in temporal times, which corresponds to 15 minutes in real world scenario. When green energy is abundant, it acts like a non-adaptive approach by activating mode 2 in this region, resulting high amount of request failure for both workloads. As the availability of the green energy decreases, the system regain responsiveness around 86th minute (see Figure 6.10). However, the controller performs poorly having 95th percentile response time of 19.79 seconds and 3.95 seconds and failed request percentage of 15% and 2% for wikipedia and fifa workload

[4]We have kept request timeout to 32 seconds
[5]The threshold value could be chosen any value between 0 and 1

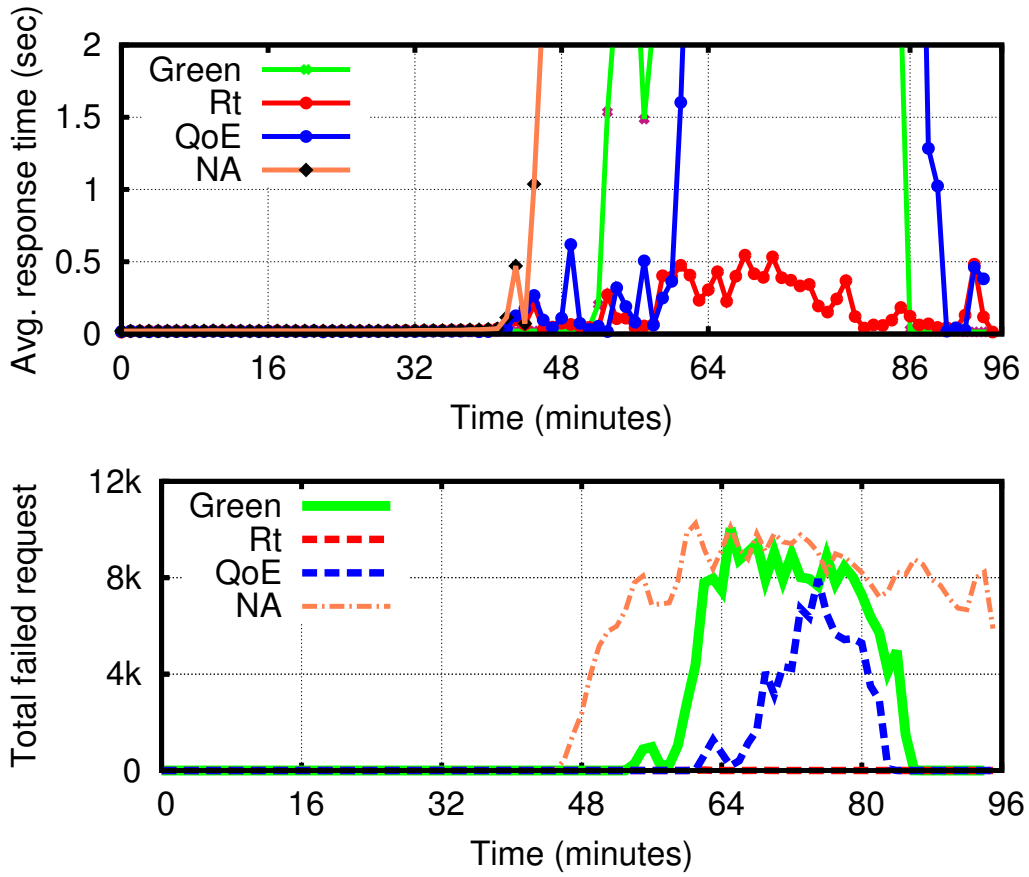


Figure 6.10: Single metric controller's performance (wikipedia workload)

respectively.

Similar to the former controller, the *QoE controller* is not aware of the response time in reconfiguration plan. As a result, the 95th percentile response time is around 11.11 seconds for wikipedia workload. But it shows better performance in terms of availability by serving 95% of total request, although Figure 6.10 indicates that the response time was beyond 2 seconds for same period of time compared to the *Green controller*. The reason being that the control time is small for the *QoE controller*; 20 seconds compared to 60 seconds. Irrespective to the load in the system, if the controller meets its target, as example: 2 recommendations for 30% of total request, the controller switches the next interval mode from *mode 2* to *mode 1* or to *mode 0*. Therefore, less number of requests wait in the web servers queue during the lower modes, since db server has to process lesser cpu intensive queries per request. Thus, the web servers receive the synchronous responses more rapidly from db tier, resulting lesser timeouts. Same behavior was observed while experimenting with fifa workload,

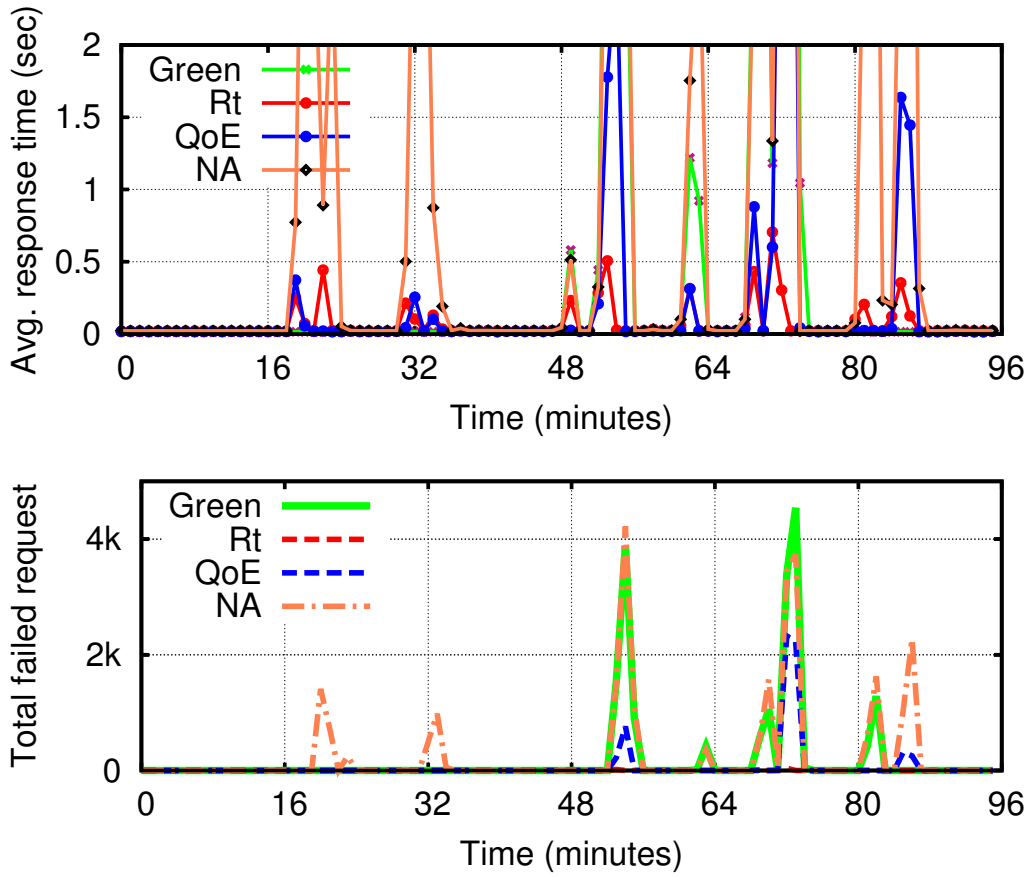


Figure 6.11: Single metric controller's performance (fifa workload)

which is showed in Figure 6.11 and 95th percentile response time is 1.36 seconds that is calculated from Figure 6.12.

6.5.5.2 Quality of experience

Figure 6.13 plots the result of SLA validation. To validate our result, we set a goal of providing recommendation to 80% requests among all the user arrives in the system and multiple recommendations at least 30% of the time, out of the 80% of the requests. Since non-adaptive approach is always activated with multiple recommendations, all the successful requests were provided with multiple recommendations, for instance, 71% of the requests. Nevertheless, serving a request with multiple recommendations and having a response time of 16 seconds, is not going to enhance a user's experience with the service. On the other hand, the *Response time controller* keeps highest user experience mode, until the system's 95th percentile response time grows beyond the set point. Figure

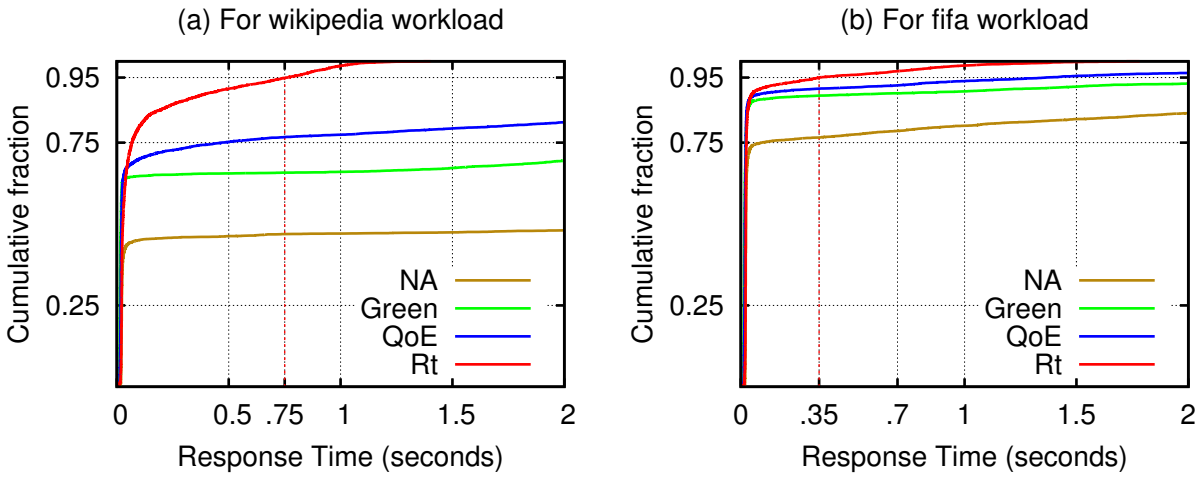


Figure 6.12: Response time in percentiles.

6.13 (a) shows that, with the increase of the workload, the controller starts disabling the recommendation around the 48th minute. Since the workload remains steadily very high for next 32 minutes, the controller detects the workload change and activates medium to low user experience mode to keep the service being saturated. As a result, 67% of the requests received recommendation on average in the whole experiment. As the number of requests increased after 20th minute in fifa workload (see Figure 6.13 (b)), the controller lower down the user experiences in the presence of workload burst and were able to provide recommendation to 76.55% of the requests.

In contrast, the *QoE controller* was designed to provide steady state percentage of recommendation in every control period. After several runs of the experiments, we found that the controller keeps the percentage of the recommendation very close to the set point (80%) by attaining 79.84% and 79.74%, and failing 5% and 1% of the requests for wikipedia and fifa workload respectively. We observed that, for fifa workload profile, all the controllers perform very closely, since the peaks are very short to destabilize the system.

Since the *Green controller* only provides recommendation when green energy is available, it can not satisfy the goal of providing 80% recommendation if the user requests is low in the green energy availability period. For wikipedia workload, relatively higher user request region belongs to the green energy period which is mid-day to late afternoon. In contrast, fifa workload has several high spikes of user requests when green energy is not available. Results show that the green controller provides 77.5% and 59% recommendation on average, which justifies our aforementioned statement.

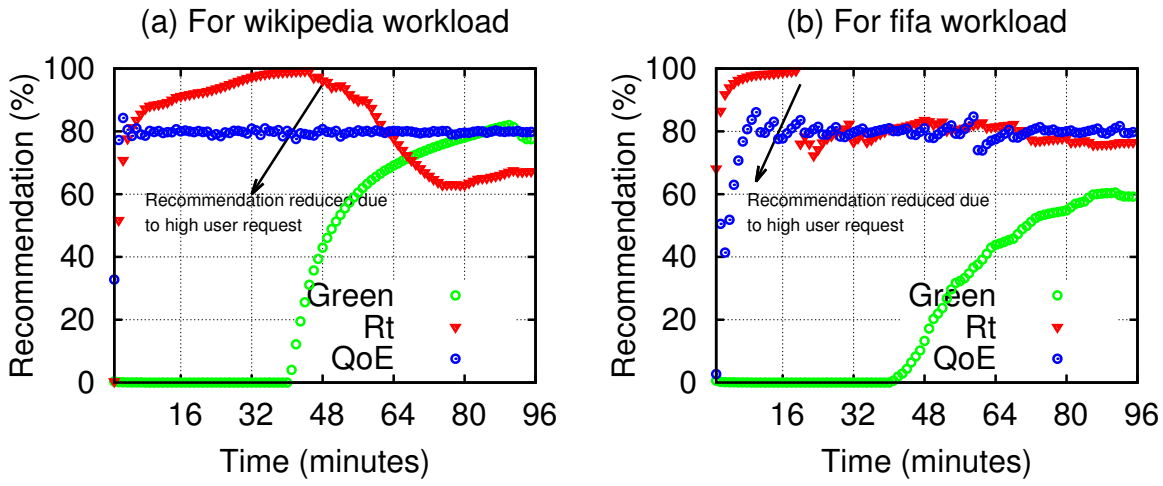


Figure 6.13: SLA validation

6.5.5.3 Energy consumption

In our experiment, each 4 minutes were considered as an hour, thus we calculated the energy consumption of 24 hours, impacted by each controller, which is presented in Table 6.1. Each experiment was run several times and we found the energy consumption difference between each run was 1~2 watts. Table 6.1 illustrates that the *Green* and *QoE controllers* were able to reduce brown energy consumption by 7.44% and 4.75% respectively for wikipedia workload. Since the *green controller* has never activated recommendation components at the period of no green energy, energy consumption was reduced in brown energy period. In addition, multiple recommendations were disabled when target SLA was met for QoE controller. As a result, reduction of brown energy was possible. Because the *response time controller* accepted and processed more requests than non-adaptive approach, it consumed 2.09% more brown energy. On the contrary, if we look at the green energy consumption by different controllers, the increment of the consumption is not very significant, which seems counter-intuitive. For instance, we investigated this phenomena for green controller and noticed oscillation of cpu consumption during 54th to 86th minute (see Figure 6.14). At the beginning of 54th minute, all the requests went from web server to db server having multiple recommendation queries. As these queries are cpu intensive and with the high number of requests, the db server quickly reached to its resource capacity even the max connection was not reached. In the mean time, the worker process of web server cannot do anything else and the queue of waiting requests grows big, even if there are more system resources available and some requests in the queue could utilize those resources. Therefore, more connections wait to connect and to receive responses from db server, causing timeout of large amount of existing requests. During these occurrences, the cpu consumption

oscillates periodically from high to low and vice-versa[6] in web server, causing domino effect in db server as well. Consequently, the aperiodic system resource consumption affects the power consumption over time, resulting in low energy consumption. Additionally, the same phenomenon was observed for the *QoE controller*.

For fifa workload, *green*, *response time* and *QoE* controllerS reduced the usage of brown energy by 13.35%, 5.59% and 6.14% respectively.

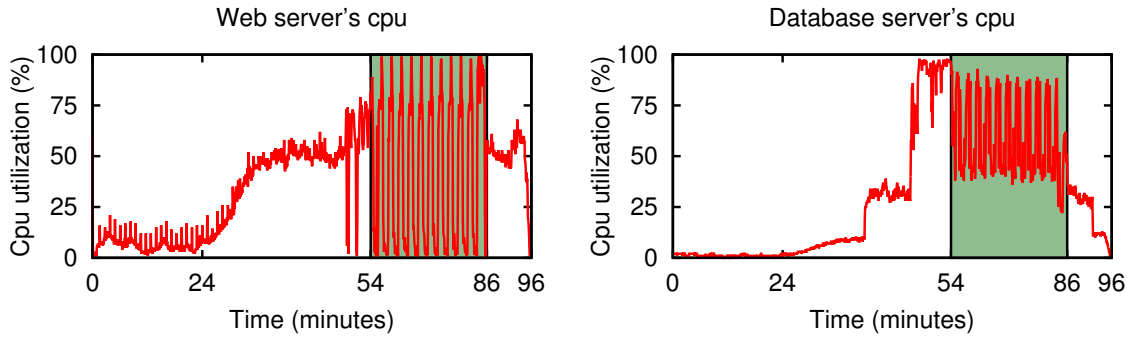


Figure 6.14: Resource consumption by Green controller

6.6 Multi-criteria controller design

In Sections 6.4 and 6.5, we elaborately explained and validated three different single metric controllers to pursue different functional and non-functional goals. It is well understood that, none of the controller can validate multiple goals *i.e.*, performance, smart usage of green energy, SLA etc. To this, we design two hybrid controllers using nested control loop architecture. The following subsections describe the characteristics, working principles, analysis and validation of two hybrid controllers.

6.6.1 Green Energy aware hybrid controller (Hybrid-green)

While response time controller can avoid performance degradation by keeping response time to a target set point, we can not guarantee of reducing energy consumption when green energy production is scarce or taking advantage of abundant green energy production. On the other hand, green energy controller can not satisfy reasonable QoS while workload arrival is high because the controller is unaware of application's internal. To this, we design a controller for considering both performance (response time) and resource aware metrics

[6]High number of requests waiting in the queue which causes timeout, that leaves the queue empty and then again filled up with new requests

Table 6.1: Energy consumption results (Wh)

Controller's name	Wikipedia workload				Fifa workload			
	Green E.C.	Brown E.C	Total E.C.	B.E.Reduction	Green E.C.	Brown E.C	Total E.C.	B.E.Reduction
Non-adaptive	1484.54	1934.66	3424.20	–	1446.01	1941.32	3387.33	–
Green	1538.78	1790.69	3329.47	7.44%	1415.92	1682	3097.92	13.35%
Rt	1518.75	1975.18	3493.93	-2.09%	1405.22	1832.62	3237.84	5.59%
QoE	1498.12	1842.72	3340.84	4.75%	1387.58	1821.99	3209.57	6.14%
Hybrid-green	1510.11	1760.09	3270.20	9.02%	1400.34	1679.53	3079.87	13.48%
Hybrid-qoe	1497.64	1909.75	3407.39	1.28%	1380.36	1797.14	3177.50	7.42%

(green energy usage) that envisages applications internal and green energy production/green energy event shown in Figure 6.15.

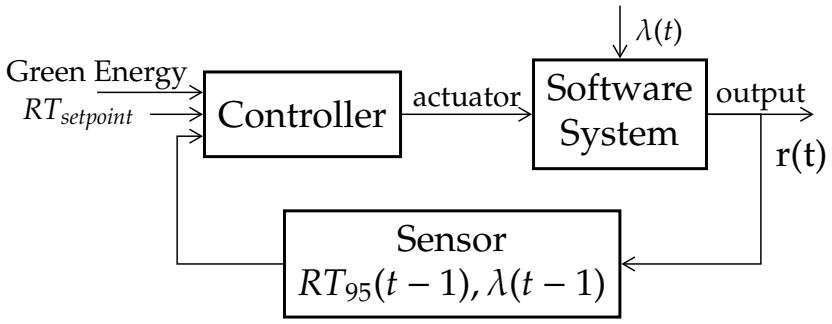


Figure 6.15: Green energy aware Hybrid controller

We distinguish between two control periods: long and short. Since green energy production does not abruptly changes so as the events that is interlinked to the presence or absence of the green energy in the data center in shorter period of time, the controller activates higher or lower user experience mode based on the event/energy information pushed by IaaS in the longer control periods *i.e.,* 15 minutes. In contrast, the controller checks the response time periodically in shorter control interval *i.e.,* 5 minutes to identify overloaded condition in the system. If occurred, the controller downgrades the user experience by subtracting 1 from previous control period's decision value (see lines 13-21 of Algorithm 22); as a result it ensures responsiveness of the application. To outline, we have merged the *green* and the *response time controllers* into a hybrid controller which has 2 different control periods or monitoring windows. Furthermore, the green part of the controller is responsible to choose the suitable modes based on the green energy availability and response time part only takes decision if the 95th percentile response time of the last control period was above the set point.

6.6.2 QoE aware hybrid controller (Hybrid-qoe)

Without considering the systems' performance, quality of experience can not be augmented since the foremost criteria of a cloud based interactive applications is to be functional and responsive even at extreme conditions *e.g.,* unexpected workload peaks, heavy workload in big duration, etc. That being said, the response time can play a critical role to keep the application responsive, while the *QoE controller* can try to maximize user experience by providing the optional contents *i.e.,* recommendations. To achieve this goal, we close the system in a nested loop manner, as showed in Figure 6.16. The outer loop provides feedback of number of request served in different modes in last control period. Whereas, the inner loop checks the 95th percentile response time and workload changes in a control period. Moreover, the inner loop is activated in shorter control period than the outer

Algorithme 7 : Green energy aware hybrid controller

Input : Thr_{max} = Threshold for green energy, λ = [0 0 0 0] = Queue to store workload arrival rate, *setPoint* = Target set point for response time, *event* = <"insufficient","ideal", "overabundance">, $Curr_{GE}$ = Current green energy production.
Output : updated λ, $Curr_{mode}$ = Current application mode.

```
1  /*Initiates in longer control period */
2  if (handleEvent == greenEnergy) then
3      if event == "insufficient" or Curr_GE == 0 then
4          app.mode ← mode 0
5      else if event == "overabundance" or Curr_GE > Thr_max then
6          app.mode ← mode 2
7      else
8          app.mode ← mode 1
9      Curr_mode = app.mode
10 return Curr_mode
11 /* Initiates in shorter control period */
12 if (handleEvent == responseTime) then
13     λ(t − 1) ← servedRequest
14     enqueue(λ)
15     function ← 1 − (λ(t − 1)/λ_median) ∗ (RT_95/setPoint)
16     if (function ≤ 0) ∧ (Curr_mode ≠ 0) then
17         app.mode ← Curr_mode − 1
18     else
19         app.mode ← Curr_mode
20     Curr_mode = app.mode
21     dequeue(λ)
22 return λ, Curr_mode
```

loop is, because the former loop can provide faster system dynamics in terms of avoiding overloaded condition. Therefore, in every big control interval, decision is taken based on the current recommendation/optional content percentage to the target SLA, as shown in Algorithm 8 (line 3-19). If it is lower than the target, higher user experience mode is activated, otherwise not. Similar to the controller mentioned in previous subsection, the inner loop downgrades user experience to the lower mode if the response time arise above the set point, if not, it periodically checks the response time without taking any decision.

6.6.3 Results

In this section, we show and analyze through our experiments how multiple metric controllers can respect their goals and if they can out perform single metric controllers or not. Furthermore, we provide a thorough discussion on cost analysis and the impact on results when the experiments are scaled in terms of using more number of servers.

Algorithme 8 : QoE aware controller

Input : Map M, Thr_1 = Target percentage for recommendation 1, Thr_2 = Target percentage for recommendation 2, $setPoint$ = Target set point for response time, $Dist_1 \leftarrow 0$ = Difference between target and current recommendation1 percentage, $Dist_2 \leftarrow 0$ = Difference between target and current recommendation2 percentage, λ = [0 0 0 0] = Queue to store workload arrival rate.

Output : updated λ, $Curr_{mode}$ = Current application mode.

```
1  /*Initiates in longer control period */
2  if (handleEvent == QoE) then
3      Total_Per1 = TotalReq_mode1/Total_req
4      Total_Per2 = TotalReq_mode2/Total_req   // Number of requests in each mode is stored in a file
       in each iteration to update and compute Total_Per1 and Total_Per2
5      update(Dist_1, Dist_2)
6      if (Total_Per1 > Thr_1) ∧ (Total_Per2 < Thr_2) then
7          app.mode ← mode 2
8      else if (Total_Per1 < Thr_1) ∧ (Total_Per2 > Thr_2) then
9          app.mode ← mode 1
10     else if (Total_Per1 < Thr_1) ∧ (Total_Per2 < Thr_2) then
11         if Dist_1 < Dist_2 then
12             app.mode ← mode 2
13         else
14             app.mode ← mode 1
15     else
16         app.mode ← mode 0
17     Curr_mode = app.mode
18 return Curr_mode
19 /*Initiates in shorter control period */
20 if (handleEvent == responseTime) then
21     λ(t − 1) ← servedRequest
22     enqueue(λ)
23     function ← 1 − (λ(t − 1)/λ_median) ∗ (RT_95/setPoint)
24     if (function ≤ 0) ∧ (Curr_mode ≠ 0) then
25         app.mode ← Curr_mode − 1
26     else
27         app.mode ← Curr_mode
28     Curr_mode = app.mode
29     dequeue(λ)
30 return λ, Curr_mode
```

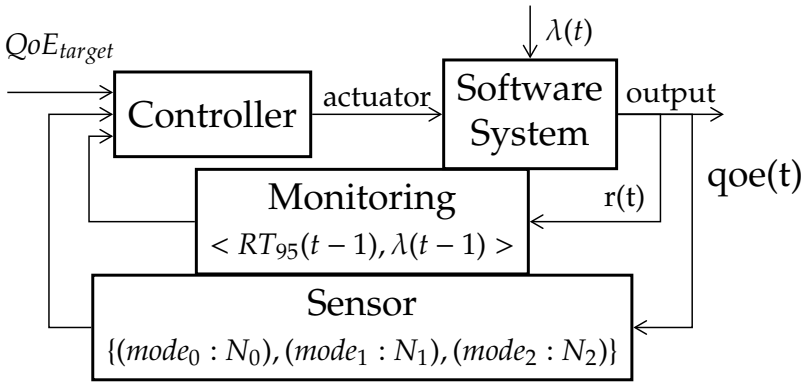


Figure 6.16: QoE aware Hybrid controller

6.6.3.1 Algorithm Implementation

From Section 6.6.1 and 6.6.2, we see that, both the controllers have inner and outer loops which are activated in different time-scales and push events to the controller to make decision. In our experiments, outer and inner loops are activated in each 60 seconds and 20 seconds respectively, which is showed in Figure 6.17. Ideally, if both kind of events arrive without any delay, two different events will overlap each other. As our motivation is to maximize of green energy usage for Hybrid-green controller, we always make primary decision based on the green energy event pushed by IaaS by ignoring the response time event which is activated as inner loop, if both the event arrives concurrently. Same phenomena applies for Hybrid-qoe controller. Concretely, it suggests that, between two big decision events in 60 seconds, we consider only two inner loop events and take actions if it is necessary indicated in Figure 6.17(a).

But in case of delaying of any event, specially for Hybrid-qoe controller where both qoe and response time metrics are pushed in a serialized manner from logstash, the scenario will not follow Figure 6.17(a). As discussed before, the original decision always depends on green energy event (Hybrid-green) and quality of experience (Hybrid-qoe) event. Even though we receive response time event, no action is taken unless the system's response is high. Therefore, in case of delaying of response time event by micro to milli seconds, effects to the system remain almost unchangeable. In contrast, if the event delays by couple of seconds, for instance, inner loop event arrives just before or after the primary decision is made, it might affect the system dynamics to achieve the goal. To tackle the problem, we define a safety distance, denoted by δ^t to ensure that the controller does not take any action if response time event arrives in between "*PrimaryDecision* - δ^t" and "*PrimaryDecision* + δ^t". Figure 6.17(b) illustrates the phenomena by an example. For our case, we choose safety distance as, δ^t = Time frequency of inner loop / 2, which is equal to 10 seconds in our experiments.

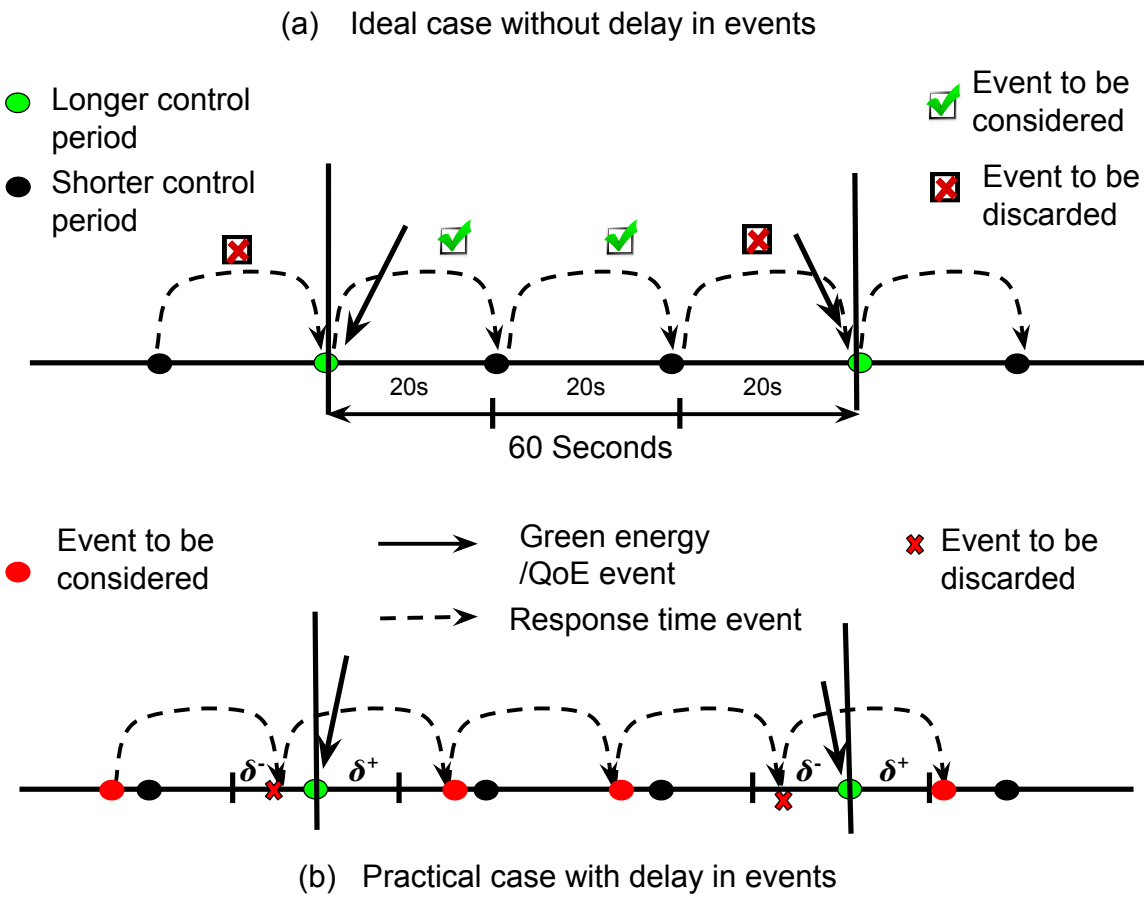


Figure 6.17: Algorithm implementation in detail

6.6.3.2 Response time

Figure 6.18 shows the average response time and number of failed request grouped by minute, caused by *hybrid-green* and *hybrid-qoe* controller. While the green controller (see Section 6.4.1) performed poorly by allowing response time to go beyond 2 seconds continuously between 56 to 84 minutes for wikipedia workload profile, *hybrid-green* controller provided better stability to the system by keeping the 95th percentile response time around 1.76 seconds (see Figure 6.20), that is 11 fold reduction of response time. However, around 60-62 minutes in the experiment, response time stayed around 2.5 seconds on average. Moreover, out of 1.7 million requests which were injected, only 6000 requests failed on average.

Additionally, the *hybrid-qoe* controller performs similarly by keeping 95th percentile

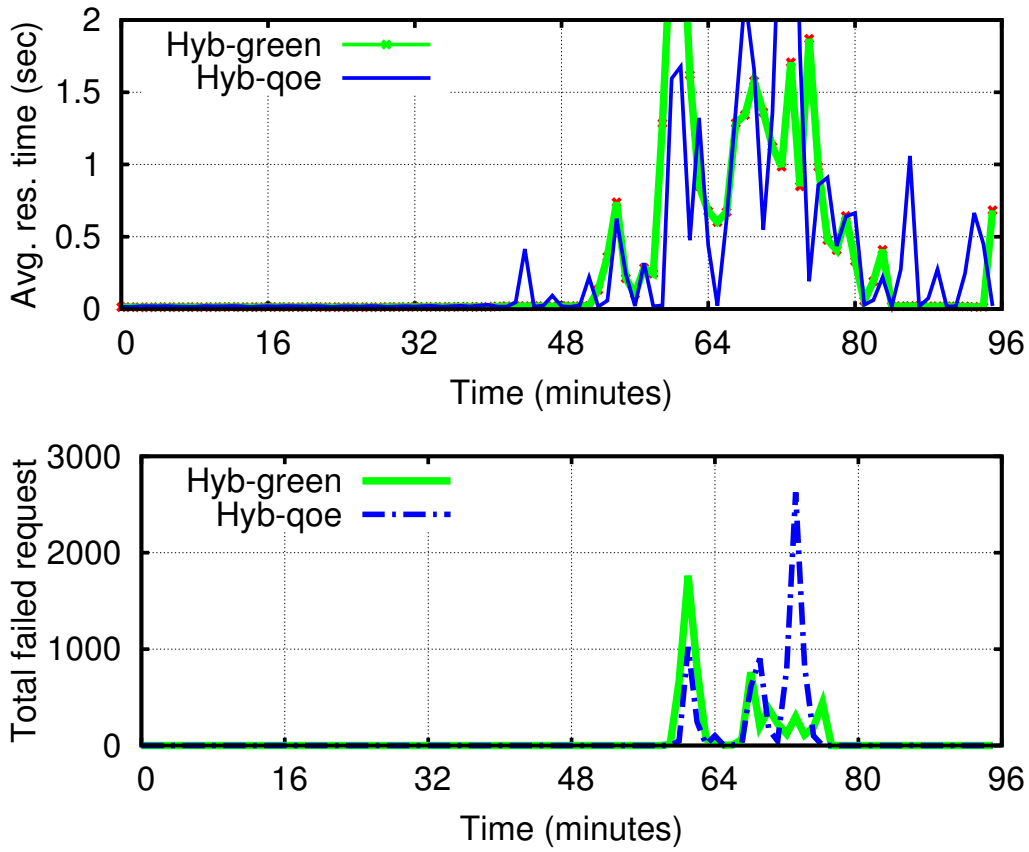


Figure 6.18: Hybrid controller's performance (wikipedia workload)

response time to around 1.93 seconds. The top curve of Figure 6.18 shows that, response time was above 2 seconds (on average 5-7 seconds) at the time of 73rd and 74th minute, causing more requests failed by *hybrid-green* controller in that region. Overall, 7530 requests failed, which again ensures 99% availability of the service. For fifa workload, both the controller foster better performance by keeping 95th percentile response time in the range of .76-1.04 seconds. During 73rd and 74th minute, average response time for *hybrid-green* controller arose to 6 seconds, causing failure of 5000 requests on average due to very high peak. Compared to former controller, 1700 requests failed on average for *hybrid-qoe* controller, which is showed in Figure 6.19.

6.6.3.3 Quality of experience

Although the QoE-based controller was able to abide the SLA, we realized that the controller was able to provide the targeted SLA *i.e.,* 79.84% for wikipedia workload only

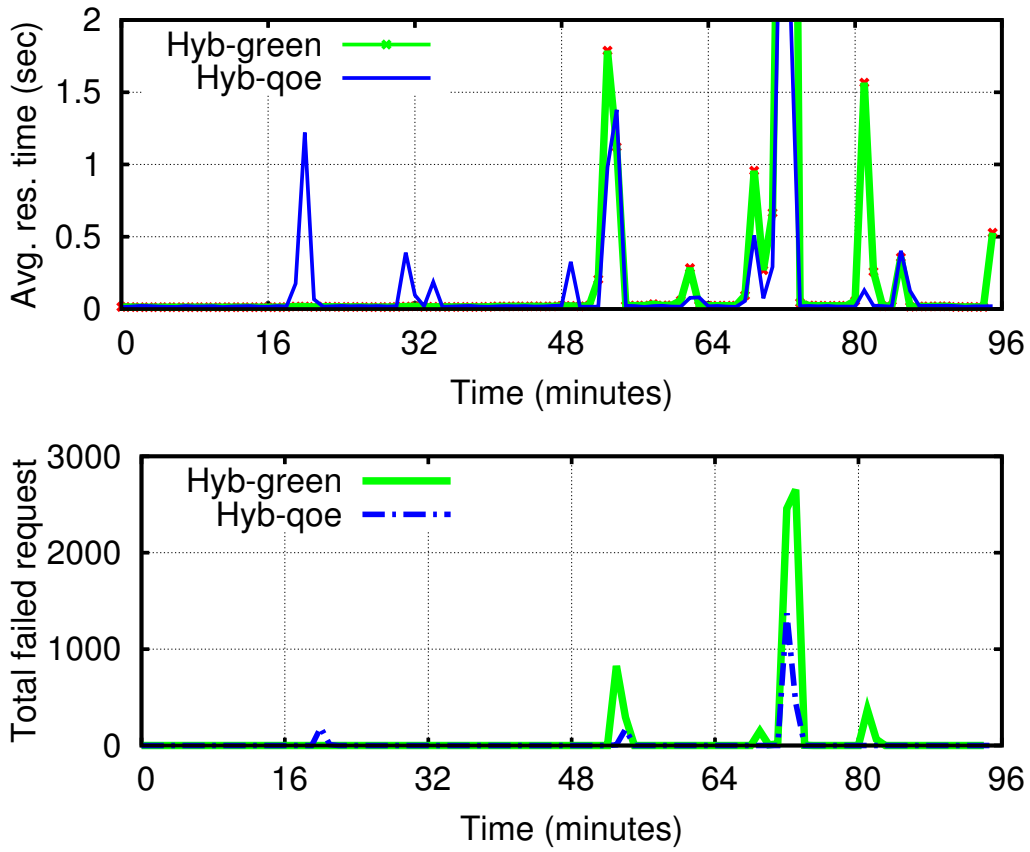


Figure 6.19: Hybrid controller's performance (fifa workload)

for successful requests. Since 5% requests failed, the provided SLA percentage might not be true oftentimes. Figure 6.21(a) shows that the *hybrid-qoe* controller can attain around 76.18% SLA while keeping number of failed requests below 1%. Although the target is missed by 3-4%, the system can accept more requests with reduced response time, providing better performance and increasing the profit for service provider. Since *hybrid-green* controller is not designed to achieve the targeted SLA, we wanted to validate how far this controller can attain the SLA. Figure 6.21(a) indicates that, on average, it can reach to 68.82% irrespective to any goal. For the like of fifa workload, where temporal peaks appears in regular intervals, *hybrid-qoe* controller performs even better by providing 79.27% recommendations on average. Adding the response time controller as an inner loop in the auto-scaler helped to detect the workload peaks. As a result, the gradient of accepted and successful request went higher than its counter-part controllers. In contrast, *hybrid-green* controller can maximize of providing recommendations only if the workload

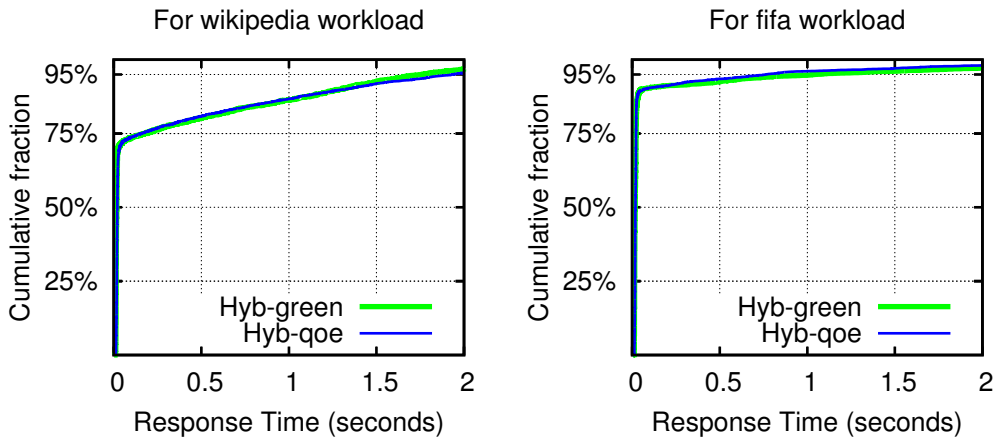


Figure 6.20: Hybrid controller's response time in percentiles

is moderate to high in the available green energy period. Since the fifa workload is not constantly high in that region, *hybrid-green* controller was able to provide only 53.21% of recommendations, as indicated by Figure 6.21 (b).

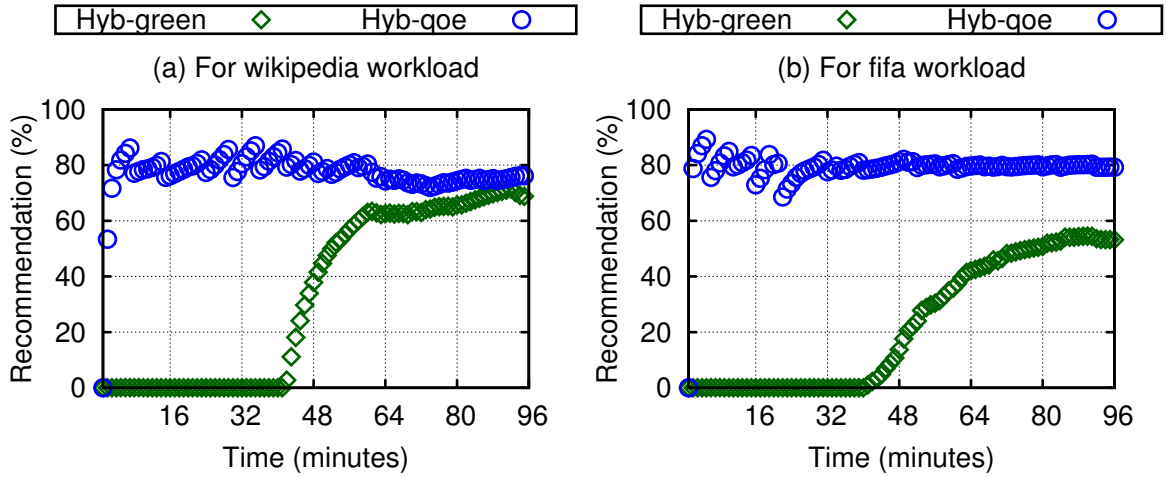


Figure 6.21: SLA validation for hybrid controller's

6.6.3.4 Energy Consumption

Hybrid-green controller reduces brown energy consumption by 9.02% and 13.48% for wikipedia and fifa workload respectively compared to non-adaptive approach, which is presented in Table 6.1. Although the *hybrid-qoe* controller lacks the capability to exploit

energy information for decision making, for wikipedia and fifa workload, it reduced fewer percentages of brown energy consumption *i.e.,* 1.28% and 7.42% accordingly. On the other hand, Table 6.1 shows that the green energy consumption was lower for both controllers. Because, system resources were not wasted for rotten requests as it happened with all the single metric controller, except for response time controller. We believe that, by reducing brown energy consumption, the carboon footprint can also be reduced. To further investigate, we propose an energy consumption metric called *Average Energy Consumption per Request (AECpR)*, which is defined as the ratio of total energy consumption over total successful requests been served. Following are the summarized result:

- For wikipedia workload, *hybrid-green* and *hybrid-qoe* controllers reduce 32.63% and 29.37% of energy consumption per request compared to non-adaptive approach.
- For fifa workload, *hybrid-green* and *hybrid-qoe* controller reduce 10.60% and 6.12% of energy consumption per request compared to non-adaptive approach.

6.6.3.5 Cost analysis

As discussed before, the requests which have been served with recommendations but have failed to keep response time under 2s might not satisfy customers, *e.g.,* customers may leave before getting the response from the system. Thus, we followed a similar approach to the one presented in [FHB10] and breakdown the monetary units to calculate provider's revenue. Each served request without recommendation corresponds to 1 monetary unit, with 1 recommendation to .25 unit, with 2 recommendations to .5 unit only if those requests were served below 2s response time and deducted 1 unit for failed requests. Figure 6.22(a) shows that, response time controller increases revenue by a big margin of 82.63% for wikipedia workload compared to non-adaptive approach and outperforms all the other controllers as well. In contrast, *hybrid-green* and *hybrid-qoe* controller decline revenue by 10.17% and 8.98% respectively compared to the response time controller. Although the revenue have decreased for two hybrid controllers, they still surpass the revenue by 64.04% and 66.21% compared to non-adaptive approach, whereas the number of requests that failed to make an impact in the revenue decreased by 53.09% and 41.44%.

Likewise, the *response time controller* increases the revenue by 7.70% in comparison with the non-adaptive approach for fifa workload. On the contrary, the non-adaptive approach can generate more revenue by 7.47% and 1.7% for fifa like workload (see Figure 6.22(b)), but *hybrid-green* and *hybrid-qoe* controllers can reduce revenue less requests by 89.5% and 88.16%. Although the non-adaptive approach can provoke to believe of having higher revenue the requests which were dropped and faced very high response time may degrade the reputation of the service provider. As a result, the gradient of revenue will decline for saturating the application in the high workload period.

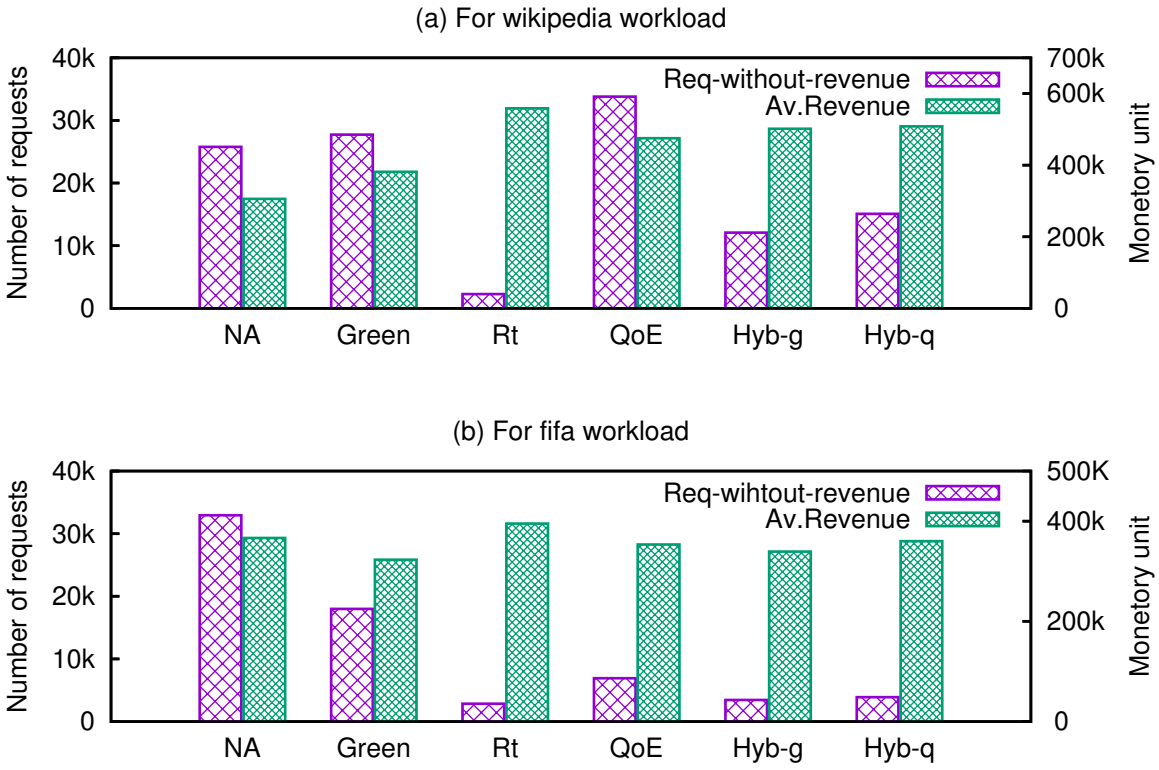


Figure 6.22: Revenue analysis incurred by all controllers

6.6.3.6 Scaled experiment

Since all the experiments were performed using a single compute node, we wanted to validate whether the energy consumption results scale as we scale the experiments or not. To do that, we performed experiment with *hybrid-green* controller by deploying three RUBiS application having wikipedia workload as a traffic pattern in three compute nodes. Theoretically, three nodes should consume around three times more compared to a single node. After doing extensive experiments, we found that, brown energy consumption deviated by as little as .02% in comparison with theoretical consumption which reflects in Figure 6.23. Furthermore, highest deviation for total energy consumption we found was .07%. With the same testbed, we performed experiment with non-adaptive approach. On average, *hybrid-green* controller could save 553.47 watts/hour of energy consumption compared to non-adaptive approach. Although the controller was not designed to provide targeted recommendation, we wanted to validate how much deviation occurs in terms of providing recommendation when we scale the experiment. The experiment validated that,

on average, the recommendation percentage deviated by 2%. To conclude, we verify that, as we scale our approach, the energy consumption increases linearly with the number of compute nodes used.

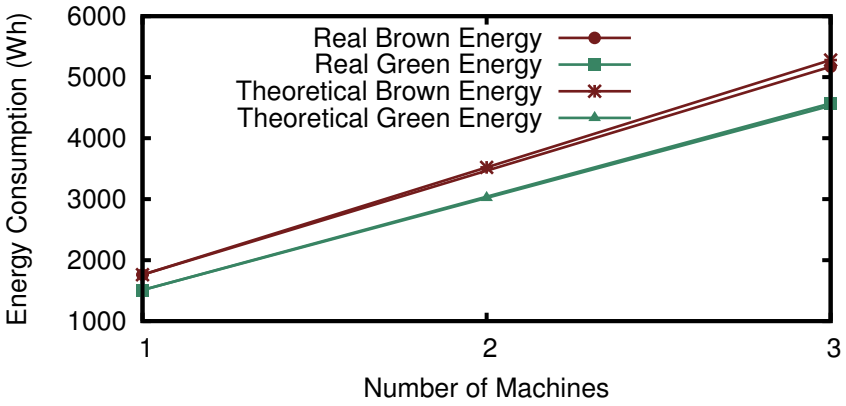


Figure 6.23: Scalability result for Hybrid-green controller

6.6.4 Discussion

After extensively analyzing all the application controllers that we have designed, it's evident that the hybrid controllers can outperform the single metric controllers in terms of energy reduction and performance gains. Table 6.2 summarizes the characteristics of different controllers to better understand the differences and impacts they can impose on the application. Nevertheless, to build efficient application controller, investigating single metric controllers is necessary to understand the relationship between different metrics. Unlike batch kind of cloud application, interactive application cannot be scheduled in advance with the presence or absence of green energy. Therefore, to create green energy awareness around cloud application, smartly using energy in the presence of green energy is the efficient way to go. Moreover, the *hybrid-green* controller can improve availability up to 15% and 29% compared to only following green energy curve and the non-adaptive approach, while the 95th percentile response time can be improved by 11% and 17% respectively for wikipedia and 4% and 6.8% for fifa workload. Hence, SaaS providers can take advantage of this controller to propose new class of SLA to eco-friendly customer who are willing to involve in reducing energy consumption. On the other hand, service providers can adopt the *hybrid-qoe* controller that can enhance user experience in higher percentage while keeping energy consumption at lower level than traditional non-adaptive approaches. In contrast, the *Response time controller* fosters highest quality of service guarantees with big margin of revenue by 82.63% when compared to non-adaptive approach. Given these trade-offs, service providers can choose any controller or switch among them based on

their goals, needs or even running conditions.

Controller's name	QoS (Response time)	QoE (Recommen-dation)	Green Energy (awareness)	Brown Energy (reduction)
Non-adaptive	✗	✗	✗	✗
Green	✗	✗	✓✓	✓
Rt	✓✓	✓	✗	✗
QoE	✗	✓	✗	✓*
Hybrid-qoe	✓✓	✗*	✓✓	✓✓
Hybrid-qoe	✓✓	✓✓	✗	✓*

(✓✓) Respect and guarantee.
(✓) Respect and guarantee to some extent.
(✗) Not at all.
(✓*) If target of recommendation percentage is very high, energy reduction is not possible.
(✗*) If majority of the workload appear at green energy period, QoE can be enhanced.

Table 6.2: Summary of applications controller's characteristics

6.7 Conclusion

This chapter investigated trade-off between energy consumption and performance by designing an auto-scaler whose objective is to adapt SaaS application's internals according to different single and multiple criteria. Our auto-scaler implements several single and nested feedback control loops intended to keep one or many of those criteria under control. We validated the controllers with an extended version of RUBiS, an eBay-like web application benchmark, under real world workload traces; and OpenStack, as Cloud Infrastructure Management tool on top of the Grid'5000 infrastructure testbed. Furthermore, our study, the first of its kind, offers a comprehensive analysis of energy consumption directly linked with cloud application. Results have shown that, by carefully tuning the application the energy consumption can be reduced while performance and revenue can be maximized.

While this approach is tested with static infrastructure *i.e.*, no resources were added of removed in run-time, we want to extend the capability of our auto-scaler to adapt at the infrastructure side as well. The next chapter investigates the issue of adapting at both Cloud layers depending on the metrics we have discussed in this chapter to see the impact on total energy consumption.

Chapter 7

Towards Green energy awareness in Cloud Platform

This chapter is an ongoing investigation on how to efficiently utilize the elasticity nature of the infrastructure resources when overall resource requirement of an application is higher than the existing underlying infrastructure can handle. Actions like adding/removing resource can be done independently at the infrastructure layer based on their utilization level *i.e.,* cpu usage, memory usage etc. But, every application performs differently from one another at same cpu utilization level, specially when the resource utilization is medium to high. Therefore, coordinating the decision based on applications resource requirement or performance is the better way to devise scaling strategies. To this, firstly we propose to listen events from application to understand when to trigger scaling decision based on reactive scaling rules. Secondly, we use traditional API such as *scale-in* and *scale-out* to trigger decision based on the strategy we have devised. Later we want to validate our approach by extensive experiments and results obtained over Grid'5000 test bed.

7.1 Motivation

Chapter 6 provides insights on how to create green energy awareness around a cloud application and results based on extensive experiments which were done using static amount of resources *i.e.,* provisioning of fixed amount of resources. But in a realistic cloud environment, resource requirement might exceed currently provisioned resources. In contrast, when lesser resources are required, de-provisioning resources can help to reduce unnecessary energy consumption. Therefore, the capability to detect when resources are required/dispensable and react to it so as to keep performance at a targeted level while energy consumption can be minimized is required. Taking application reconfiguration decision in isolation with resource scaling policies may lead to performance degradation and inconsistency to the system. Hence, coordination between two different types of action

is necessary.

Most of the work in the literature proposes: (i) multiple autonomic loops in a coordinated manner to control cluster level resources (*i.e.,* one loop for controlling DVFS, another loop for deciding scaling actions)[WW11], [SDMD+16]; (ii) two autonomic loops one at application level, another at infrastructure level to adapt at both layer in a coordinated manner by formulating constraint programming model to solve resource requirement problem. [dOJL12]; (iii) per-application local manager which requests to a central autonomic manager to tune the number of cpu core, memory and to change the number of VM's [CPMK11], [CKMP17]; (iv) adaptive framework to coordinate between system level (DVFS) and application level (degrading quality) adaption to improve performance and power efficiency [HKCH16]. (v) A Domain Specific Language (DSL) called ElaScript, providing high-level support for Cloud administrators to simply and safely express reconfiguration plans orchestrating the different levels of elasticity actions [DBADOL17].

In response to the existing works, we propose a PaaS solution that inherits the capability to adapt both at application and infrastructure level in facing to changing condition. Application adaptation is realized by dynamically reconfiguring application on the fly, whereas infrastructure adaptation takes care of addition/removal of resources based on resource demand. We want to study the impact of application adaption (based on the presence/absence of renewable energy) on infrastructure to have a global view of energy consumption. Furthermore, both adaption technique is built in separate modules and coordinated in a sequential manner. For example, when application's performance decreases due to heavy load, the PaaS solution first triggers adaptation to application by downgrading the functionality and invokes resource requests to infrastructure module. Followed by the invocation requests, infrastructure adaptation module analyzes and decides whether resources are going to be added or the request is to be ignored. Following sections detail the PaaS solution along with different controllers.

7.2 GPaaScaler architecture

Chapter 6 (Section 6.3) presented our auto-scaler architecture. Compared to that, we propose an upgraded auto-scaler, named *GPaaScaler*, which inherits the capability to actuate both at application and at infrastructure level. Our contribution lies on the *analyze* and *plan* (A-P) block. Monitoring block pushes listened events to *Analyze* block from SaaS layer (*i.e.,* response time, workload, application's working mode, etc.) and IaaS layer (*i.e.,* quality of energy). We then analyze and decouple events to extract the information and feed appropriate event to the event handler at the SaaS controller. Once the configuration plan is ready, SaaS controller triggers action through SaaS actuator and passes request for addition/removal of resources event as « RequestEvent » to IaaS controller if the former controller decides that application needs more/less resources, which is shown at Figure 7.1. Following the event, IaaS controller decides to take action via traditional infrastructure

API that is *scale-in* and *scale-out* or wait/discard the request issued by the SaaS controller. Therefore, the execution block is composed of two types of actuators, which can be seen at Figure 7.1. The sequential flow of the event in an ordered way (from 1.*a* to 1.*e*) is shown at the Figure 7.1 as well to understand at ease. In summary, IaaS controller only gets activated if SaaS controller issues any « RequestEvent ». However, our proposed IaaS controller are unware of resource allocation strategy, for instance, what types of VM is to be added/removed or in which server VM is to be located etc.

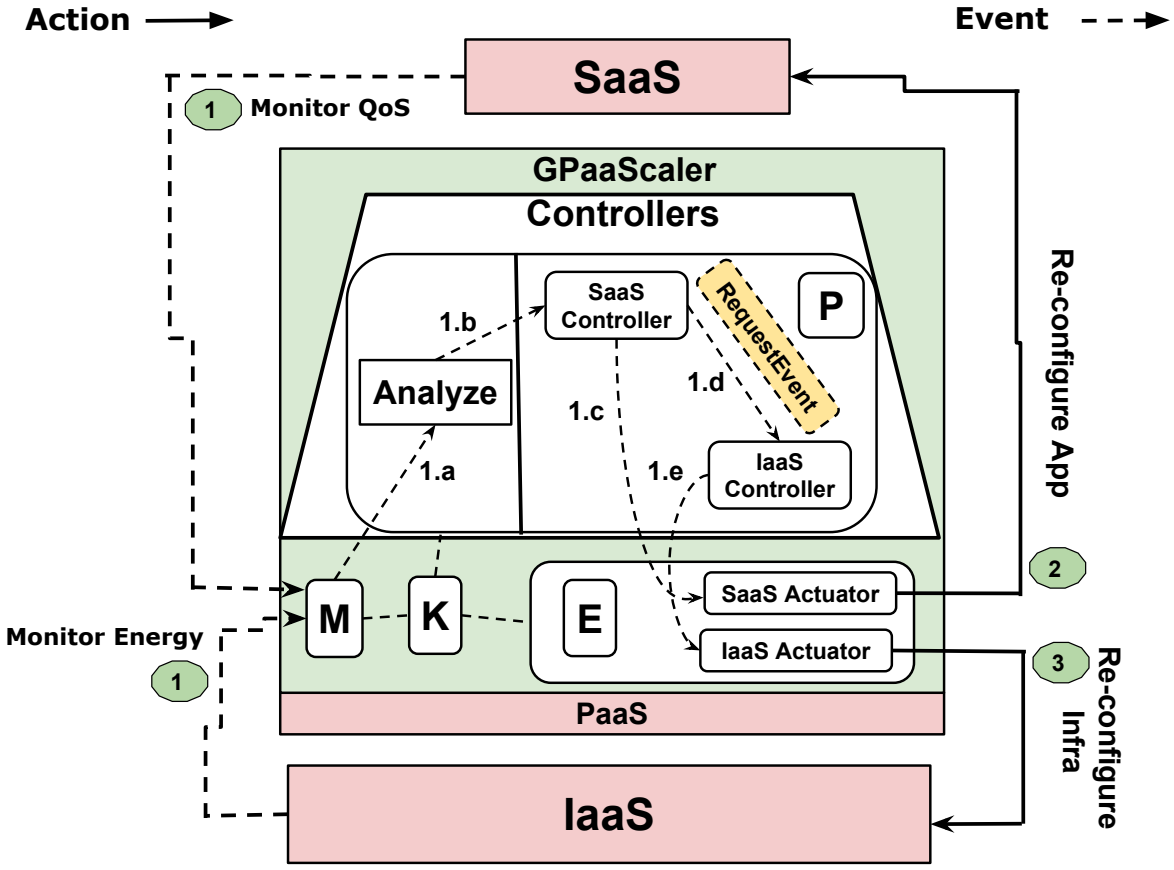


Figure 7.1: GPaaScaler architecture

7.3 SaaS controllers

We have designed and validated several single and mutiple metric application controllers which have the capability to re-configure SaaS application to keep it accessible and

performant even at changing conditions at Chapter 6. In this section, we extend the Green Energy aware hybrid controller (quality of resource aware) and Response time controller (performance aware) with increased capability to request of addition/removal of resources to the infrastructure layer.

Extended Green-hybrid controller (EGH-C). In this extended version of the controller, we try to investigate when performance indicator of an Application can trigger add/remove VM request. Since this controller have two feedback loops activating at two different control period: long and short, and longer control period's decision depends only on the energy information, hence we rather investigate the shorter control period loop which is based on response time event. The shorter control loop periodically checks if the performance of the application is degraded or not (*i.e.,* violating targeted response time) by computing a function at line 15 at Algorithm 26. If the computed function becomes negative (function ≤ 0) meaning, if the current response time is beyond or borderline to set point and/or the tendency of the workload is increasing, the controller downgrades the user experience by subtracting 1 from previous control period's decision value and notify a `vmAdd` event request to the infrastructure controller (see line 18, Algorithm 26). While the function is greater than 0, which suggests that the application is performing well by keeping current 95th percentile response time to the set point, application keeps the user experience as before but notify a `vmRemove` event request to the infrastructure controller (see line 21, Algorithm 26). In both cases, « RequestEvent » notifies the specific event along with application's current 95th percentile response time and workload increment ratio to the IaaS controller.

Extended Response Time controller (ERT-C). At Chapter 6 : Section 6.4.2, we have presented a response time controller and validated as a most performant controller in terms of respecting QoS properties compared to other application controllers at Section 6.5.5. Therefore, we decided to extend this controller so that it can take advantage of underlying elastic infrastructure. To do that, we compute a function to analyze how far the multiplication ratio of workload and response time increment/decrement is from 1. If the function is positive and above a desired/predefined threshold, the controller keeps the highest user experience mode *i.e.,* mode 2. Since the controller is not aware of how much amount of underlying resources are used, it notifies the *vmRemove* event to the IaaS controller (see line 7, Algorithm 10). In case, the condition block falls to *function* ≤ 0, a « RequestEvent » of *vmAdd* is notified to IaaS controller (see line 10, Algorithm 10).

7.4 IaaS controller

While under-provisioning of resources can significantly hamper QoS properties by saturating application, over-provisioning of resources can increase energy consumption

Algorithme 9 : Extended Green-hybrid controller

Input : Thr_{max} = Threshold for green energy, λ = [0 0 0 0] = Queue to store workload arrival rate, *setPoint* = Target set point for response time, *event* = <"insufficient","ideal", "overabundance">, $Curr_{GE}$ = Current green energy production.
Output : updated λ, $Curr_{mode}$ = Current application mode.

```
1  /* Initiates in longer control period */
2  if (handleEvent == greenEnergy) then
3      if event == "insufficient" or Curr_GE == 0 then
4          app.mode ← mode 0
5      else if event == "overabundance" or Curr_GE > Thr_max then
6          app.mode ← mode 2
7      else
8          app.mode ← mode 1
9      Curr_mode = app.mode
10 return Curr_mode
11 /* Initiates in shorter control period */
12 if (handleEvent == responseTime) then
13     λ(t − 1) ← servedRequest
14     enqueue(λ)
15     function ← 1 − (λ(t − 1)/λ_median) ∗ (RT_95/setPoint)
16     if (function ≤ 0) and (Curr_mode ≠ 0) then
17         app.mode ← Curr_mode − 1
18         RequestEvent → vmAdd /* VM Addition request event sent to IaaS controller along
           with RT_95 and workload-increment = (λ(t − 1)/λ_median) */
19     else if (function > 0) then
20         app.mode ← Curr_mode
21         RequestEvent → vmRemove      /* VM Removal request event sent to IaaS controller
           along with RT_95 and workload-increment = (λ(t − 1)/λ_median) */
22     else
23         app.mode ← Curr_mode
24     Curr_mode = app.mode
25     dequeue(λ)
26 return λ, Curr_mode
```

significantly. Therefore, the scaling decision, for instance, add resources (scale-out) or remove resources (scale-in) should be taken carefully to match with the applications resource demand. To meet *scale-out* condition, a reactive policy can be easily designed and implemented based on the monitored performance metrics or listening to predefined appropriate events. A reactive policy is referred to a run-time decision based on current demand and system state - to add resources on the fly. On the contrary, reactive policies can not absorb the non-negligible resource/instance initiation time. In our case, when applications starts to face high response time, the SaaS controllers have the capability to

Algorithme 10 : Extended Response time aware controller

Input : $Thr_{rt}, \lambda = [0 \quad 0 \quad 0 \quad 0], setPoint, app$
Output : updated λ, $Curr_{mode}$ = Current application mode.

```
1  if (handleEvent == responseTime) then
2      λ(t − 1) ← servedRequest
3      enqueue(λ)
4      function ← 1 − (λ(t − 1)/λ_median) ∗ (RT_95/setPoint)
5      if (function > 0) ∧ (function < Thr_rt) then
6          app.mode ← mode 1
7          RequestEvent → vmAdd /* VM Addition request event sent to IaaS controller along
           with RT_95 and workload-increment = (λ(t − 1)/λ_median) */
8      else if function ≤ 0 then
9          app.mode ← mode 0
10         RequestEvent → vmAdd /* VM Addition request event sent to IaaS controller along
           with RT_95 and workload-increment = (λ(t − 1)/λ_median) */
11     else
12         app.mode ← mode 2
13         RequestEvent → vmRemove      /* VM Removal request event sent to IaaS controller
           along with RT_95 and workload-increment = (λ(t − 1)/λ_median) */
14     dequeue(λ)
15     Curr_mode = app.mode
16 return λ, Curr_mode
```

degrade the user experience level and to invoke an implicit event (`vmAdd`) request to IaaS controller. Therefore, the sequential operation can trigger the application to run at lower mode until the instance is launched and activated. Afterwards, the application revert back to higher mode if it meets the condition after operation.

In contrast, when *scale-in* event (*i.e.*, fewer resources are required by application) is invoked by SaaS controllers, terminating instance based on reactive policy can have detrimental impact on the system. For example, when application performs better by staying just below or borderline to set point, triggering *scale-in* action can make an application suffering from high response time to saturation. One way to overcome the problem is to reduce the number of cpu cores[1] on the fly by doing fine-grained analysis of resource requirement rather than terminating an entire instance, but popular hypervisors like KVM, VMware, Hyper-V does not allow removing cpu cores of guest VMs at run-time [TL14]. Additionally, instance termination can cause a sharp rise in response time reaching beyond the set point if workload's behavior or tendency is not taken into consideration. Therefore, devising a plan when to execute *scale-in* event is critical. On the other hand, if the consecutive scaling actions are carried out too quickly without being able to observe the impact of scaling action to the application, undesirable effects such as over and under-

[1] In case, CPU is the bottleneck in the application

provisioning of resources can occur which can leads to performance degradation and/or wastage of energy consumption.

Algorithme 11 : Infrastructure controller

Input : $[minVm, maxVm]$ = Minimum and maximum number of VM's.
$[RT_{95}, workload_{inc}]$ = Response time and workload increment sent by SaaS controller.
$[rt_{thr}, decWorkPerc]$ = Two tunable parameters.
Output : *vmNumber*, *coolingPeriod*

```
1  if (handleEvent == vmAdd) then
2      if (currentTime ∉ coolingPeriod) ∧ (vmNumber < maxVm) then
3          triggerAction → "scale − out"        /* Passing API call through cloud infrastructure
           manager */
4          vmNumber+ = 1
5          coolingPeriod+ = coolingLength
6      else
7          vmNumber = this.vmNumber
8          coolingPeriod = this.coolingPeriod
9      vmNumber = update(vmNumber)
10     coolingPeriod = update(coolingPeriod)
11 return vmNumber, coolingPeriod
12 if (handleEvent == vmRemove) then
13     if (currentTime ∉ coolingPeriod) ∧ (rt_thr > RT_95) ∧ (vmNumber > minVm) ∧
14     [(workload_inc < decWorkPerc ∨ Curr_mode = 0)] then
15     triggerAction → "scale − in"     /* Passing API call through cloud infrastructure manager
       */
16     vmNumber− = 1
17     coolingPeriod+ = coolingLength
18     else
19         vmNumber = this.vmNumber
20         coolingPeriod = this.coolingPeriod
21     vmNumber = update(vmNumber)
22     coolingPeriod = update(coolingPeriod)
23 return vmNumber, coolingPeriod
```

Hence, the idea is to built a generic IaaS controller which is characteristically agnostic to SaaS controllers behavior. Whenever, an implicit event invocation (`vmAdd, vmRemove`) arrives to the controller, it activates the proper module matching to the event. Since, two nonconcurrent events can be invoked by SaaS controllers, our proposed IaaS controller contains two modules to handle each of them. We define a length of period called *coolingLength*, which is composed of instance activation time and the time it requires to impact on the application. Therefore, after triggering any scaling decision, this time period is updated to prevent any scaling decision to be made in between. Hence, when `vmAdd` event arrives to the controller, the *handleEvent == vmAdd* module matches the condition of

not being at *coolingPeriod* with an and operator to maximum number of VM a provider can be assigned to[2]. If it adheres the condition, *scale-out* decision is triggered via IaaS actuator and current number of VM and next *coolingPeriod* is updated (see line 3-5 of Algorithm 11). Otherwise, the module ignores the notification. On the other hand, when `vmRemove` event invokes by SaaS controller, if the *handleEvent* == *vmRemove* module is not carefully designed, cloud application can face unstable phases *i.e.*, sharp rises of response time to saturate application. Therefore, only looking at *coolingPeriod* and minimum number of VM[3] could be unwise and skeptical. Therefore, we introduce two key parameters which are tunable to identify when is the good time to release resources *i.e.*, perform scale-in action. The parameters are i) how far the current system's response time should be from set point? For example, x% less than target response time set point, which is denoted by rt_{thr} at Algorithm 11. ii) how much workload should decrease from the current trend? For instance, y% decrease in user request than previous intervals, denoted by *decWorkPerc*. Hence, when *handleEvent* == *vmRemove* arrives to the IaaS controller, the module checks the cooling period, minimum number of VM, current response time condition with an AND operator. Additionally we put an OR operator between workload decrease parameter and current mode of the application. The rationale behind that, in the absence of green energy, *EHG-C* keeps the application at minimum level. Although, workload may be consistent or increasing, if the hybrid controller satisfies being outside of *coolingPeriod*, greater than minimum number of VM and reduced response time that the threshold, it will meet the *scale-in* condition and IaaS controller will trigger the action to release resources. On the other hand, *ERT-C* will keep application at the highest mode when resources are slightly to abundantly over-provisioned. Thus, application being at *mode* = 0 and decreasing workload by y% percentage can not happen concurrently if response time is x% less than response time set point for this type of SaaS controller. Apart from *EHG-C*, any SaaS controller which invokes `vmRemove` event and satisfies all the conditions mentioned above other than application mode being at lowest, will trigger *scale-in* action by IaaS controller.

7.5 Discussion

This chapter provides insights on how to adapt at infrastructure layer depending on application performance and presence/absence of green energy in data center. In very near future, we would like to validate the algorithms that we have designed. Currently we are experimenting at Grid'5000 test bed and are analyzing the results. Our interest relies on investigating energy consumption incurred by each controller when they are coupled with an generic infrastructure controller, while targeted QoS properties can be met.

[2]Amazon EC2 permits maximum 20 on-demand instances per user.
[3]For a 3-tier application, at least one VM per tier should always run.

Chapter 8

Conclusion

This chapter concludes this thesis by revisiting the problem statement which has been stated throughout the document and summarizing the main contribution to highlight the effectiveness of the solution. Finally, we discuss some perspectives based on this research to point out future directions.

8.1 Problem Statement Revisited

Due to the proliferation and adoption of Cloud services which reside in data centers, enormous energy consumption became a critical issue. In response, existing researches are focused more on reducing energy consumption, but the goal for alleviating carbon footprint is far form the expectation. Furthermore, *greenness* of Cloud services and data centers remain questionable. One way to overcome this problem is to introduce renewable energy usage opportunities to data center by incorporating them. By nature, green energy sources are intermittent. Therefore, how to manage green energy sources at data center level to greenify the cloud infrastructure is a considerable research challenge. Hence, exploring different renewable energy integration options and different pricing is required to find an efficient energy management policy in data center to tackle the intermittent nature of green energy. On the other hand, if Cloud application can take advantage of the presence or absence of green energy to change its energy state, further energy consumption reduction is possible along with other traditional approaches. Unlike batch kind of applications, *interactive* Cloud applications can not be scheduled in advance depending on green energy profile. Scheduling plans can mismatch with run-time variations of an application caused by workload surge, resource limitations, etc. Therefore the problem can be defined as: how to make Cloud application, specially *interactive* application, adaptive to green energy availability while traditional QoS properties can be at satisfactory level so to lower carbon footprint? While creating green energy awareness can be a fruitful solution, how to manage the addition/removal of resources while application reacts to green energy events is also

a key issue to be addressed. Because, decisions taken in isolation at given layer may mismatch the resource requirements by the application, that can negatively impact QoS and energy consumption reduction goal.

8.2 Summary of Contributions

To overcome the stated problem, this thesis provides four contribution as follows:

Cloud Energy Broker. To propose green computing services powered by renewable/green energy, first we have to investigate the different green energy integration options and their advantages as well as disadvantages and procurement strategies. Therefore, Chapter 4 investigates the opportunity to exploit the energy market to plan, forecast and purchase energy in advance through a Cloud Energy Broker to greenify the data center in advance.

Virtualization of Green energy. Chapter 5 introduces Virtualization of green energy concept to tackle both the forecasting error of Chapter 4 and intermittency of green energy to propose and revise the notion of GreenSLA. The idea is to propose new class of explicit SLO mentioning the percentage of green energy provided along side with computing service by managing the underlying energy infrastructure and multi-source energy market. GreenSLA gives the possibility to application owners to host their application in an explicitly expressed green cloud environment having formal contracts. Moreover, virtualization concept has leverage over energy storages in terms of energy loss or wastage. Therefore, we provided an efficient energy management plan which can maximize the usage of green energy in data center by disallowing any energy wastage.

Green energy awareness in application. Chapter 6 presents a self-adaptive autoscaler architecture to enable smart usage of energy in an interactive application. The autoscaler inherits the capability of sensing information as events from multiple layers while actions are performed only in application level. Based on the autoscaler architecture, we devised several application controller to satisfy different metrics of interest and validated through extensive experiments at Grid'5000. Thus, the proposed contribution can make an application adaptive by automatically adjusting to changing conditions, while respecting QoS properties.

Towards Green energy awareness in platform. Chapter 7 investigates how to efficiently utilize the elasticity nature of the infrastructure resources while reconfiguration capability at application level can be coordinated for better performance and reduced energy consumption. Towards the goal, we propose GPaaScaler architecture which can both actuate at application and at infrastructure level depending on the application behavior

and energy availability. Therefore, the platform become aware of green energy availability/unavailability. Later, we design a generic infrastructure controller which can be used with any application controller by invoking additional/removal resource requests. The results indicate that, when the coordination between application and infrastructure are done efficiently, energy reduction is possible with targeted performance.

8.3 Perspective

This thesis work have investigated trade-off between energy management, energy consumption and performance while proposed solution to tackle those problems. Some of the proposal presented as contribution can lead to interesting and promising research perspectives. We provide some discussion on following subsection.

8.3.1 Selection of VM types based on fine-grained resource demand

In this thesis, we analyze when an application requires additional/lesser resources and take actions accordingly by taking the advantage of underlying infrastructure. While our proposed solution can help to decrease performance degradation, more fine grained analysis and estimation of resource demand can leads to select proper VM types. When resource demand slightly increases to cause performance degradation of an application, a large VM can help to stabilize the system but may cause over-provisioning of resources if demand remains steady. On the other hand, when resource demand increases abruptly, adding a tiny VM will not suffice and performance degradation can prompt to create several tiny VMs sequentially. In contrast, during the decrement of resource demand, releasing a proper VM type would be necessary from preventing a chain reaction of adding and releasing VMs. Apart from performance constraint, application owner also have budget constraints to reduce service cost. When application owners rent VM's from infrastructure provider, they are charged by instances/hour. Therefore, during the termination of instances, finding a set of proper VM type along with which VM's are closer to their instance hour can be a noteworthy investigation. Thus an application will be able to guarantee performance with lesser energy consumption while service cost can remain under control.

8.3.2 Containerized approach

In this thesis, we have applied our solution on virtual machine environment which is enabled by hardware level virtualization. In recent years, Operating System (OS) level virtualization that virtualizes resources at OS level is rapidly increasing, hence *container* technology has gained much attraction. Due to the low-overhead at virtualization level, containers can provide better throughput and lesser latency for I/O intensive interactive application [SCJSC16]. Additionally, containers resume quite faster typically in the range of seconds, while VM takes couple of minutes to be accessible. Due to the hardware virtualization,

each VM is runs in top of a guest operating system. Hence, this virtualization overhead may increase the energy consumption while running same workload on containers. By default, a container has no resource constraints (soft resource limits) like VM's (During VM boot up, its hard limit is initialized), thus it can use as much resources as the host kernel allows. This may prevent the system to take less coordinated action between application and infrastructure compared to VM approach. Therefore, we think it will be an interesting track of investigation to analyze how much energy efficient containers are against VMs.

8.3.3 Leveraging Microservice architecture for application adaptation

In recent years, modern cloud application has moved from monolithic approach to a distributed approach, where application is refactored to small units, each providing a single functionality. All small units communicate with each other in a synchronous or asynchronous manner. In this thesis, for the use case, we dynamically adapted application by adding/removing recommendation component on the fly. By using microservice architecture, recommendation components can be composed and deployed as a separate unit. By doing that, when system load increases, they can be either replicated or scaled in an autonomic way on different computation nodes. Apart from the example of recommendation component, any resource hungry application component can be seen as a separate entity and be deployed in similar fashion. Since, containers can be scaled in matter of few seconds, it would be interesting to deploy small and decoupled elements throughout different containers so that, each of the application component if required, can be scaled to guarantee better performance, self healing capabilities. Apart from that, resource can be assigned to specific components where it is required, thus over-provisioning of resource phenomena can be avoided resulting lesser energy consumption. Additionally, this investigation can leads to a decentralized autonomic behavior in modern application. Today's popular application like Netflix[1], Groupon[2], etc. have moved to adopt microservices architecture recently. Therefore, we believe that, this area of research investigation can open the door of automating fully decentralized cloud application.

8.3.4 From Cloud to Fog/Edge computing

With the advent of Fog/Edge computing, data computation and fetching can be done near to the users. It is done through deploying small mono-site data centers consisting of 50-100 servers in multiple sites so to leverage on data computation in a faster way locally rather than forwarding to core data center and fetching the results, which can be time sensitive. Furthermore, each mono-site data center can be powered by different renewable sources causing different renewable energy profile in these data centers. While in this thesis, we only consider single data center where an application can be adaptive to green

[1] https://www.netflix.com/
[2] https://www.groupon.com/

energy production. This idea can be easily extended to multiple mono-site data centers. One idea could be, putting an global autonomic agent to gather information from local autonomic managers at data center level *i.e., green energy production* and at application level *i.e.,* response time, user experience level, etc. Afterwards, we can formulate a plan to route requests in a way to consume more available energy across the data centers, while performance and user experience can be guaranteed and enhanced. Furthermore, it will be also interesting to investigate a trade-off of how much processing needs to be done between Edge cloud and Core Cloud in terms of performance and energy consumption.

Appendices

Scientific Production

1. **Journal Article**

   (a) **Md Sabbir Hasan**, Frederico Alvares de Oliveira, Thomas Ledoux, and Jean Louis Pazat. "Investigating Energy consumption and Performance trade-off for Interactive Cloud Application", *IEEE Transactions on Sustainable Computing (T-SUSC)*

   (b) Nicolas Beldiceanu, Barbara Dumas Feris, Philippe Gravey, **Md Sabbir Hasan**, Claude Jard, Thomas Ledoux, Yunbo Li, Didier Lime, Gilles Madi-Wamba, Jean-Marc Menaud, Pascal Morel, Michel Morvan, Marie-Laure Moulinard, Anne-Cécile Orgerie, Jean-Louis Pazat, Olivier Roux, Ammar Sharaiha. "Towards energy-proportional Clouds partially powered by renewable energy". *Computing, Springer*, vol. 99, pp 3-22, January 2017.

   (c) **Md Sabbir Hasan**, Yousri Kouki, Thomas Ledoux, and Jean Louis Pazat. "Exploiting Renewable sources: when Green SLA becomes a possible reality in Cloud computing", *IEEE Transactions on Cloud Computing (TCC)*, vol. PP, Issue 99, July, 2015.

2. **Conference Article**

   (a) **Md Sabbir Hasan**, Frederico Alvares de Oliveira, Thomas Ledoux, and Jean Louis Pazat. "Enabling Green Energy awareness in Interactive Cloud Application", *In Proceedings of the 8th IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, December 12-15, Luxembourg, 2016.

   (b) Nicolas Beldiceanu, Barbara Dumas Feris, Philippe Gravey, **Md Sabbir Hasan**, Claude Jard, Thomas Ledoux, Yunbo Li, Didier Lime, Gilles Madi-Wamba, Jean-Marc Menaud, Pascal Morel, Michel Morvan, Marie-Laure Moulinard, Anne-Cécile Orgerie, Jean-Louis Pazat, Olivier Roux and Ammar Sharaiha. "The EPOC project: Energy Proportional and Opportunistic Computing system." *In Proceedings of the 4th International Conference on Smart Cities and Green ICT Systems (SMARTGREENS)*, Lisbon, Portugal, May 20-22, 2015.

(c) Yousri Kouki, **Md Sabbir Hasan** and Thomas Ledoux. "Delta Scaling: How Resources Scalability/Termination Can Be Taken Place Economically?" *In Proceedings of the IEEE World Congress on Services (SERVICES)*, June 27 - July 2, New york, 2015.

(d) **Md Sabbir Hasan**, Yousri Kouki, Thomas Ledoux, and Jean Louis Pazat. "Cloud Energy Broker: Towards SLA-driven Green Energy Planning for IaaS Providers", *In Proceedings of the 16th IEEE International Conference on High Performance Computing and Communications (HPCC)*, August 20-22, Paris, 2014.

Résumé en Français

Conséquence directe de la popularité croissante d'Internet et du Cloud, les centres de données de petite à grande taille sont en plein essor. En 2007, les centres de données en Europe de l'Ouest ont consommé 56 terawatt-heures (TWh) de puissance par an. Selon l'UE, ce chiffre devrait presque doubler pour atteindre 104 TWh d'ici à 2020. La même année, selon le Gartner group, l'industrie des ICT a représenté 2% des émissions mondiales de carbone. Ces émissions de carbone élevées sont le résultat de la production d'électricité à partir de combustibles fossiles ou de charbon. Bien que la France génère 75% de l'électricité par les centrales nucléaires qui émettent relativement moins de carbone, la quantité d'empreinte carbone n'est nulle part proche de zéro. En outre, les coûts énergétiques liés à la puissance opérationnelle d'un centre de données représentent 15% du coût global de possession (TCO).

L'une des principales raisons de la croissance de la consommation d'énergie est que de plus en plus de fournisseurs de services transfèrent leur application ainsi que leur charge de travail informatique vers le Cloud. En 2016, Rightscale a signalé que l'adoption des services en nuage a augmenté de 13-14% par rapport à 2015. La première raison de cette migration vers le Cloud est de diminuer les coûts liés aux technologies de l'information et d'être opérationnel sans investir lourdement ou maintenir leur propre infrastructure informatique. De plus, les fournisseurs de logiciels en tant que service (SaaS) ou les propriétaires d'applications cherchent à garantir un certain niveau de performance et la disponibilité de leurs services, a.k.a des applications interactives Cloud sans aucune perturbation pour les utilisateurs finaux. Pour garantir les conditions de qualité de service (QoS), les applications Cloud doivent toujours fonctionner et être réactives indépendamment de la suggestion du trafic utilisateur aussi, les fournisseurs d'infrastructure en tant que service (IaaS) doivent sur-provisionner les ressources, mais les fournisseurs SaaS sont prêts à payer uniquement ce qu'ils consomment. En revanche, les fournisseurs d'IaaS visent à faire tourner des machines physiques de moindre envergure non seulement pour réduire le TCO, mais aussi pour réduire la consommation d'énergie et leur empreinte associée. Bien que le TCO puisse être diminué par le biais d'un sous-provisionnement de ressources, il peut dégrader les propriétés QoS de l'application hébergée. Par conséquent, les objectifs contradictoires dans différentes couches de service sont gérés par Service Level Agreement (SLA), c'est-à-dire une description formelle des contraintes temporelles, de performance et économiques entre

SaaS et fournisseurs IaaS.

Énoncé du problème

Alors que la prolifération des services Cloud qui résident dans les centres de données a un impact important sur notre société, la *greenitude* de leur nature reste discutable. Cette *greenitude* peut être basée sur des techniques d'efficacité énergétique et sur la qualité de l'énergie consommée dans ces centres de données. En réponse, les recherches existantes se concentrent plus sur la réduction de la consommation d'énergie en concevant/implémentant la consolidation de serveurs [BAB12] [HH13], du matériel avec de meilleurs compromis puissance vs performance [VAN08] et des techniques logicielles pour l'ordonnancement conscient de l'énergie [KMAHR14], etc. Bien que ces efforts soient nécessaires, l'objectif d'atténuer l'empreinte carbone est loin d'être atteinte. Selon un rapport récent, la consommation d'électricité a augmenté d'environ 4% par rapport à 2010-2014, ce qui représente un changement important par rapport à l'augmentation de 24% estimée à partir de 2005-2010. Cependant, on s'attend à ce que la consommation d'énergie augmente continuellement avec le même taux de 4% pour 2014-2020 comme les cinq années passées. Par conséquent, les données susmentionnées indiquent que l'efficacité énergétique seule ne va pas réduire l'empreinte carbone car la consommation d'énergie continuera de croître. Au contraire, l'intégration explicite ou implicite des énergies renouvelables au centre de données peut être une mesure complémentaire pour accompagner les techniques d'efficacité énergétique et ainsi réduire l'empreinte carbone.

Problème 1. La plupart des centres de données d'aujourd'hui ne sont connectés qu'au réseau électrique où l'électricité est produite par la combustion du charbon et du gaz naturel, qui sont des approches de production d'énergie à forte intensité de carbone. Par conséquent, si le réseau électrique n'est pas respectueux de l'environnement en termes de compensation et de neutralisation de l'émission nette de carbone zéro ou de l'énergie « brune », les grands consommateurs comme les centres de données doivent se tourner vers d'autres alternatives « vertes », c'est à dire des sources d'énergies renouvelables sur site ou hors site. Le défi considérable de la recherche de l'intégration des sources d'énergie verte dans le centre de données est qu'elles sont intermittentes par nature, donc toujours pas disponibles. En dehors de cela, la plupart des publications de recherche ne sont pas conscients des analyses de coût de la nature variable des prix de l'énergie verte, jusqu'à considérer l'énergie verte pour un coût de $0. Par conséquent, explorer les différentes options d'intégration des énergies renouvelables et les prix peut conduire à une politique efficace de gestion de l'énergie dans les centres de données pour faire face à l'intermittence de l'énergie verte. Une fois le problème résolu, les fournisseurs IaaS peuvent proposer des services d'informatique orientés *green* aux consommateurs SaaS ou aux clients finaux.

Problème 2. Les applications, qui sont hébergées dans le centre de données, peuvent être sommairement classées en tant qu'applications Batch ou Interactive. Alors que les premières peuvent être considérées comme tolérants aux délais, les dernières peuvent être très sensibles aux retards, sinon la QoS peut être fortement touchée. Comme toute technique de gestion, une gestion efficace de l'énergie peut être contrainte de ne pas avoir assez d'énergie verte, qu'il s'agisse d'une installation sur place ou d'un marché de gros, en cas de catastrophe naturelle, de jours nuageux, de prix élevés de l'énergie, etc. Pour cela, plusieurs travaux ont été proposés et expérimentés sur la façon de planifier ou exécuter des applications Batch lorsque l'énergie verte est disponible afin de consommer plus d'énergie verte et ne pas en gaspiller. Il en résulte une réduction des énergies brunes dans le centre de données, ce qui a un impact positif sur l'objectif de réduction de l'empreinte carbone. Par contre, la charge de travail des applications interactives / la réponse au trafic ne peut pas être retardée et cela exclut toute planification des tâches. La question de recherche est donc de savoir comment rendre l'application SaaS interactive adaptée à la disponibilité de l'énergie verte, alors que les propriétés traditionnelles de QoS doivent rester satisfaisantes, pour réduire l'empreinte carbone. De plus, l'adaptabilité de l'énergie verte dans les applications interactives en nuage n'a pas encore été abordée dans les recherches existantes. Étant donné que les préoccupations sociétales et environnementales ont suscité des initiatives en matière d'énergie verte, il est grand temps de considérer la valeur verte de la métrique énergétique comme un attribut essentiel, parallèlement à la qualité de service traditionnelle.

Problème 3. Alors que la couche IaaS permet d'ajuster dynamiquement la fourniture de ressources physiques en fonction des besoins de la plate-forme en tant que service (PaaS) pour optimiser l'efficacité énergétique du centre de données, la réduction de l'empreinte carbone est encore insuffisante. L'intégration des différentes énergies renouvelables au niveau des centres de données et l'adoption de l'Autonomic Computing (AC) à la couche SaaS pour gérer *greenitude*, la réactivité et l'autonomie face aux changements environnementaux pourraient être une solution réalisable pour améliorer l'écosystème Cloud. De plus, dans la couche SaaS, AC peut permettre aux applications de réagir à une charge de travail très variable et à la présence d'énergie renouvelable en ajustant dynamiquement la quantité de ressources afin de maintenir la QoS pour les utilisateurs finaux. Cependant, des problèmes peuvent survenir car ces systèmes autogérés sont liés d'une certaine manière (par exemple, les applications dépendent des services fournis par une infrastructure Cloud): les décisions prises isolément à une couche donnée peuvent ne pas correspondre aux exigences de ressources de l'application, ce qui peut avoir un impact négatif sur la QoS et l'objectif de réduction de la consommation d'énergie.

Par conséquent, créer une conscience d'énergie verte dans l'application interactive et adapter intelligemment de manière auto-adaptative au contexte changeant, va être la seule façon de réduire l'empreinte carbone.

L'état de l'art

***Écologisation* de l'infrastructure Cloud computing** Dans le chapitre 3, nous présentons une sélection de travaux pertinents sur *l'écologisation* de l'environnement en nuage et des centres de données en termes de spécification SLA et la gestion de l'énergie. Nous nous abstenons de discuter et d'esquisser des technologies permettant l'efficacité énergétique (par exemple DVFS et divers états de sommeil pour les serveurs) et des techniques comme, par exemple, la migration de VM ou la consolidation de serveurs, ces méthodes étant largement adoptées et pratiquées depuis plusieurs années. Notre objectif est de chercher plus loin pour réduire l'empreinte carbone en gérant efficacement différentes sources d'énergie qui peuvent être imposées de l'utilisateur final à l'autorité de réseau. Afin d'évaluer les travaux décrits précédemment, nous résumons les efforts de recherche en définissant certains attributs qualitatifs de comparaison:

- **GreenSLA:** Comme la demande pour les produits verts est de plus en plus importantes, les utilisateurs sont plus conscients de la *greenitude* du produit, que ce soit dans le supermarché ou dans le logiciel et le matériel. Par conséquent, la spécification de SLA est nécessaire. Habituellement dans la littérature, la notion de GreenSLA est utilisée pour assouplir certaines exigences de performance pour réduire la consommation d'énergie.
- **Energy management:** En général, les fournisseurs de services Cloud effectuent la gestion de l'énergie pour réduire le coût énergétique qui représente environ 20% du coût total [GHMP08]. En réduisant les coûts, l'objectif est d'atténuer les émissions de carbone dans la phase d'exploitation. Cependant, le choix du prix et de la diversité de l'emplacement ([RLXL10], [FYH+15]) ne peut garantir une émission de carbone plus faible car le facteur d'émission du réseau peut être élevé à ce moment-là ou à cet endroit. Par conséquent, nous classons la gestion de l'énergie en fonction de la présence d'énergie brune et d'énergie mélangé (énergie brune et verte).

Après avoir analysé rigoureusement deux domaines de travail très différents, nous nous rendons compte que la gestion de l'énergie et le SLA peuvent être liés ensemble. Par conséquent, dans cette thèse, le chapitre 4 et 5 étudie la possibilité d'exploiter le marché de l'énergie tout en proposant et révisant la notion de GreenSLA. Notre idée est de proposer une nouvelle classe de SLO explicite mentionnant le pourcentage d'énergie verte fournie en même temps que le service informatique, grâce à une gestion de l'infrastructure énergétique sous-jacente et du marché de l'énergie multi-sources. Ainsi, les utilisateurs et les fournisseurs d'applications peuvent avoir la possibilité d'héberger leur application dans un environnement de nuage vert possédant des contrats formels sur l'énergie verte.

Application Cloud consciente de l'énergie et des performances La section 3.3 du chapitre 3 présente une sélection de travaux pertinents et populaires autour des applications

Cloud conscientes de l'énergie verte et de la performance. En outre, nous définissons certains attributs qui classifient les travaux susmentionnés pour mieux comprendre le problème que nous voulons aborder:

- **Conscience de l'énergie verte:** Par ce terme, nous voulons dire qu'une application est consciente de la variabilité de l'énergie verte et change la durée ou le moment d'exécution des applications en conséquence.
- **Optimisation des coûts:** La plupart des recherches sur les centres de données se concentrent sur la façon de réduire les coûts liés à l'électricité. Le coût énergétique lié au réseau électrique peut être réduit par l'utilisation opportuniste de l'énergie verte disponible, alors que certains efforts ont été faits pour exploiter l'emplacement et la diversité des prix pour une réduction supplémentaire.
- **Garantie de performance via l'adaptation:** La performance est le premier critère de QoS pour toute application en nuage, mais ceci peut varier en fonction de la nature de l'application. Par exemple, respecter l'échéance est une exigence principale pour l'application de type Batch. D'autre part, la latence et la disponibilité est la préoccupation principale pour l'application interactive.

Afin de surmonter le problème de la création d'une conscience d'énergie verte dans une application interactive en nuage, notre deuxième partie de la thèse du chapitre 6 présente une architecture auto-adaptative (grâce à l'informatique autonome) pour permettre une utilisation intelligente de l'énergie verte dans une application interactive. Notre idée est de transporter l'information énergétique à l'application afin qu'elle puisse s'adapter en fonction des événements énergétiques afin de réduire la consommation d'énergie « brune ». Nous proposons plusieurs contrôleurs d'application qui font le compromis entre la QoS, la QoE, la consommation d'énergie brune et verte, pour mettre en lumière que l'idée d'adapter intelligemment l'application avec l'information sur l'énergie est possible.

Contribution

Par conséquent, dans cette thèse, pour aborder les problèmes discutés dans la section 8.3.4, nous suivons une approche de bas en haut, de l'infrastructure à l'application. Nos contributions dans cette thèse sont les suivantes:

- **Chapitre 4.** La demande de services verts augmente considérablement au fur et à mesure que les gens prennent de plus en plus conscience de l'environnement pour bâtir une société durable. Par conséquent, les entreprises et les clients souhaitent héberger leurs services ou leurs applications dans un environnement de nuages plus vert offert par le fournisseur Infrastructure-as-a-Service (IaaS). Pour construire un environnement Cloud plus vert autour du centre de données, l'efficacité énergétique

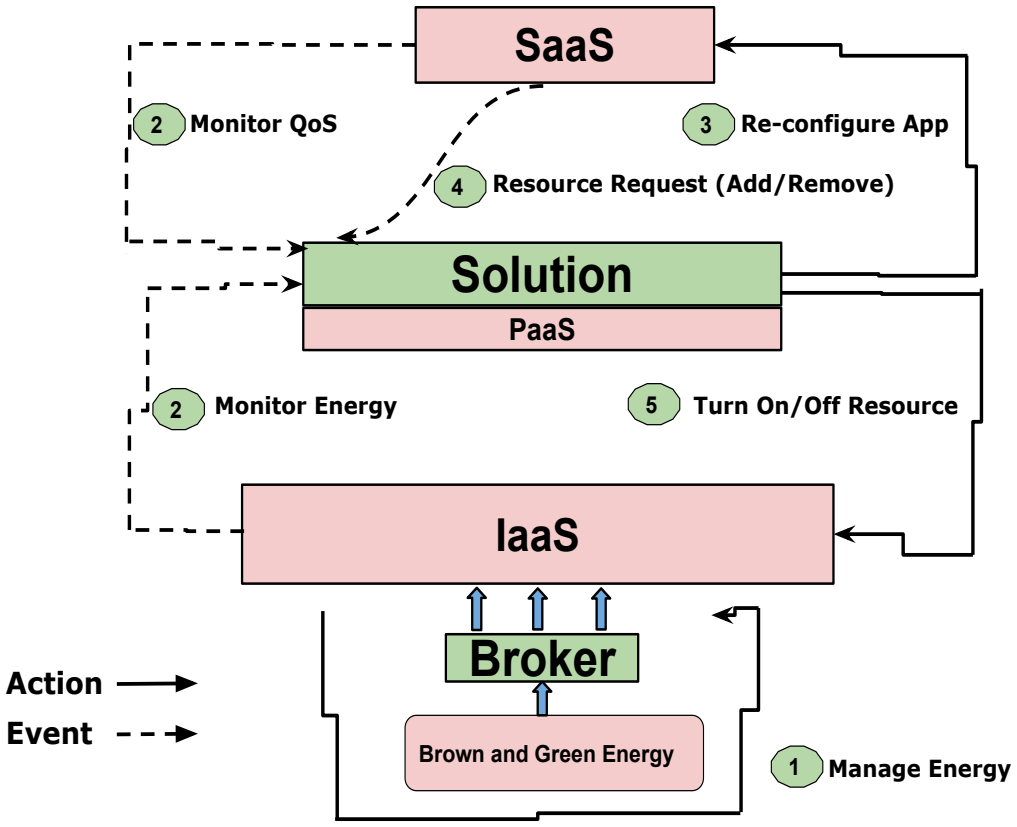


Figure 8.1: Overview of proposed solution

maximale et l'impact environnemental minimum, (c'est-à-dire une empreinte carbone inférieure) sont les critères les plus importants. Pour cela, plusieurs techniques d'efficacité énergétique pour les systèmes matériels et logiciels ont été proposées dans la littérature qui sont largement adoptées et exploités. En revanche, la gestion de l'énergie des centres de données en présence de sources implicites et explicites d'énergie verte qui peuvent faciliter la réduction de l'empreinte carbone est encore à ses balbutiements, mais elle a suscité beaucoup d'intérêt ces derniers temps. Le principal défi pour un fournisseur IaaS est de déterminer le meilleur compromis entre ses bénéfices tout en utilisant l'énergie verte avec une contrainte budgétaire et contrat de niveau de service (SLA) avec le Software-as-a-Service (SaaS) et le fournisseur d'énergie. D'autre part, pour fournir des services informatiques verts au fournisseur ou au client SaaS, une SLA robuste doit être traitée. Par conséquent, dans

ce chapitre, nous expliquons différents niveaux d'objectifs de niveau de service (SLO) entre chaque couche de nuages afin de réaliser comment la SLA à plusieurs couches peut être contractée en présence d'énergie verte dans l'environnement de Cloud computing (voir Figure 8.2). En outre, nous proposons un *Cloud energy broker*, qui peut ajuster la disponibilité et la combinaison de prix pour acheter de l'énergie verte dynamiquement à partir du marché de l'énergie à l'avance pour rendre un centre de données vert basé sur un contrat SLA. Plus tard, la validation du courtier en énergie est fournie pour montrer qu'il peut maintenir avec succès le meilleur compromis entre la disponibilité de l'énergie et la contrainte budgétaire. En outre, ce chapitre présente la phase de planification de la gestion de l'énergie verte pour les centres de données.

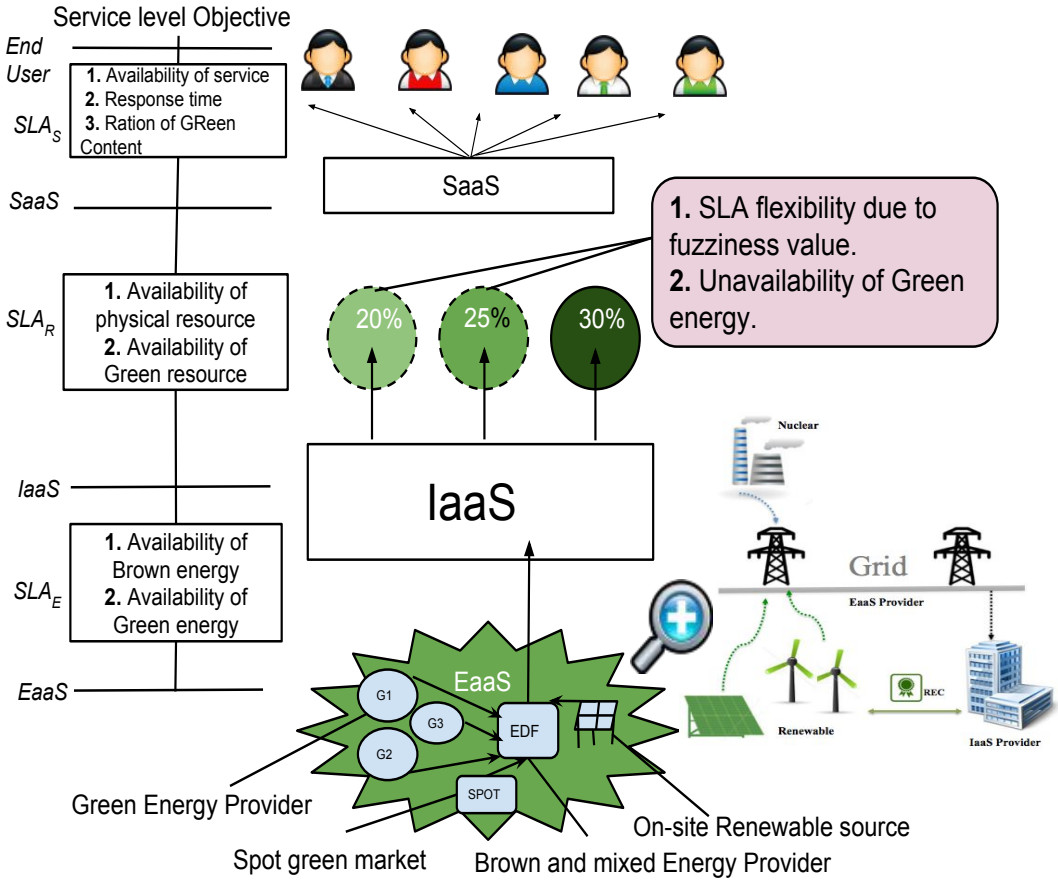
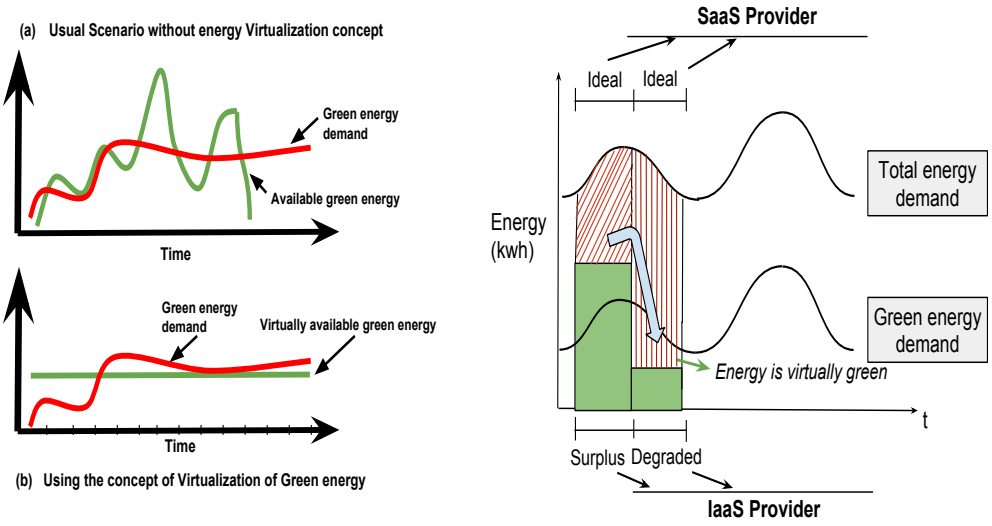


Figure 8.2: Cross-layers SLA

- **Chapitre 5.** La définition et l'établissement de *GreenSLA* entre le IaaS et le fournisseur

SaaS basé sur l'énergie verte est absent de la littérature. Par conséquent, le principal défi pour le fournisseur IaaS est de gérer les SLA verts avec leurs clients tout en satisfaisant ses objectifs métiers, tels que la maximisation des profits tout en réduisant les dépenses pour l'énergie verte. Mis à part les incitations des organismes publics et privés, le coût de production d'énergie verte et les prix sur le marché tendent à être plus élevé que l'énergie « brune ». Puisque les Green SLA doivent être proposés en fonction de la présence d'énergie verte, la nature intermittente des sources renouvelables rend la validité du contrat SLA difficile à atteindre. En réponse, ce chapitre présente un schéma de gestion de l'énergie verte en temps réel en présence d'une intégration explicite et implicite de l'énergie verte dans le centre de données. Plus précisément, nous proposons trois contributions: i) nous introduisons le concept de virtualisation de l'énergie verte (voir Figure 8.3) pour répondre à l'incertitude de la disponibilité de l'énergie verte ; ii) nous étendons le langage CSLA (Cloud Service Level Agreement) pour permettre un SLA vert en introduisant deux nouveaux paramètres de seuil et iii) nous introduisons l'algorithme de greenSLA qui exploite le concept de la virtualisation de l'énergie verte pour fournir par intervalle un SLA vert spécifique. Des expériences ont été menées avec le profil réel de charge de travail de PlanetLab et le modèle de puissance de serveur de SPECpower pour démontrer qu'un Green SLA peut être établi avec succès et sans coût supplémentaire. La Figure 8.1 illustre l'aperçu de la solution proposée et (1) indique la position de la contribution susmentionnée.



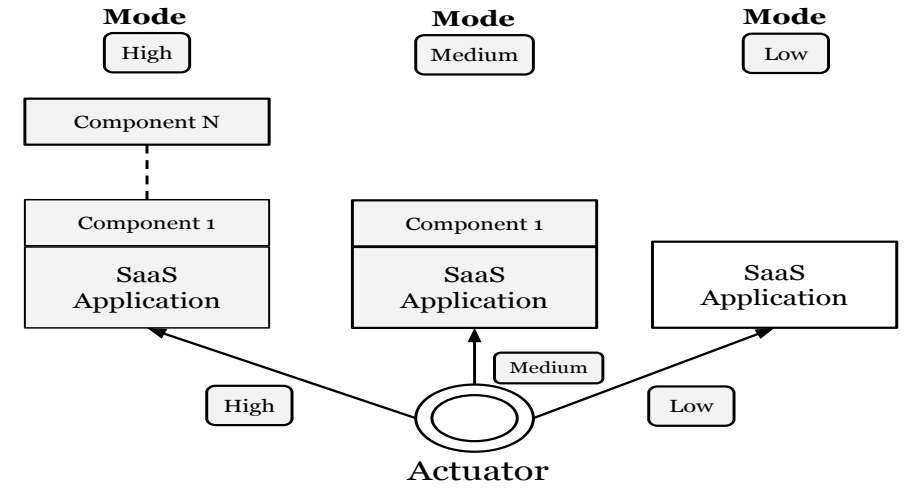
(a) Effect of green energy virtualization

(b) Validation of virtualization of energy

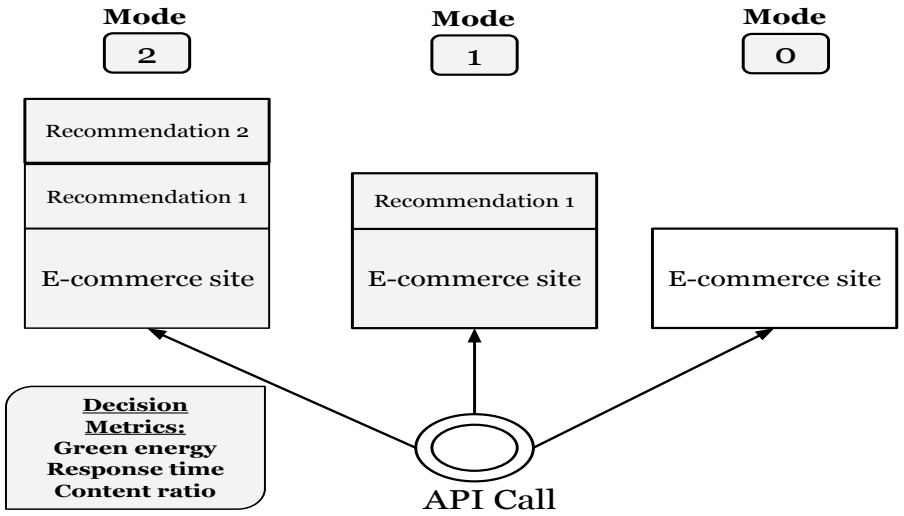
Figure 8.3: Green energy virtualization concept

- **Chapitre 6.** Avec la prolifération de l'informatique en nuage, les centres de données doivent d'urgence faire face à des problèmes de consommation d'énergie. Bien que les efforts récents tels que l'intégration des énergies renouvelables dans les centres de données ou les techniques d'efficacité énergétique dans les machines (virtuelles) contribuent à la réduction de l'empreinte carbone, l'utilisation intelligente de l'énergie verte dans les applications Cloud n'a pas encore été abordée. Par utilisation intelligente, nous entendons la prise de conscience d'une application Software-as-a-Service (SaaS) d'augmenter la consommation d'énergie pendant la disponibilité de l'énergie verte et réduire la consommation d'énergie quand l'énergie verte est rare ou absente. Dans ce chapitre, nous proposons une architecture auto-adaptative basée sur l'Autonomic Computing, qui hérite de la capacité de capter des informations en tant qu'événements à partir de couches multiples alors que les actions ne sont réalisées qu'au niveau application. Ainsi, notre approche peut rendre une application adaptative en s'adaptant automatiquement aux conditions changeantes d'exécution. Dans notre approche, l'application peut fonctionner sur différents modes en fonction du niveau de service, ce qui est montré à la Figure 8.4. En outre, nous étudions plusieurs contrôleurs d'application basés sur différentes métriques (par exemple, disponibilité d'énergie verte, temps de réponse, niveau d'expérience utilisateur). Grâce à des expériences approfondies et l'analyse sur une application réelle dans l'environnement Cloud réel, l'utilisation intelligente de l'énergie verte est validée. Nous fournissons deux contrôleurs hybrides, qui peuvent fournir des garanties formelles de certification du temps de réponse du 95e percentile des systèmes gérés à proximité de la cible, tandis que la consommation d'énergie brune peut être réduite jusqu'à 13%. En outre, notre approche ajuste également l'exigence de capacité dynamiquement en libérant des ressources virtuelles pour permettre à 29% d'utilisateurs de plus d'accéder à l'application SaaS. (2) et (3) à la Figure 8.1 illustre notre contribution dans ce chapitre. Cette partie de la contribution vise à fournir des idées et des stratégies nécessaires pour rendre l'application interactive SaaS consciente de et adaptative à l'énergie verte.

- **Chapitre 7.** Dans ce chapitre, nous étudions comment utiliser efficacement l'élasticité des ressources d'infrastructure lorsque l'exigence de ressources globale d'une application est plus élevée que l'infrastructure sous-jacente existante peut gérer. Des actions comme l'ajout/suppression de ressources peuvent être effectuées indépendamment au niveau de l'infrastructure en fonction de leur niveau d'utilisation *i.e.*, utilisation du CPU, utilisation de la mémoire, etc. Mais chaque application fonctionne différemment l'une de l'autre au même niveau d'utilisation de CPU spécialement quand l'utilisation des ressources est moyenne à élevée. Par conséquent, la meilleure façon de concevoir des stratégies de mise à l'échelle est de coordonner la décision en fonction des besoins en ressources ou de la performance des applications. Pour cela, nous proposons d'abord d'écouter les événements de l'application, qui est marqué par (4) à la figure



(a) Example of different application modes



(b) Realistic example of application modes

Figure 8.4: Application modes under different service level

1.1 pour comprendre quand déclencher la décision de passage à l'échelle basée sur des règles réactives. Deuxièmement, nous utilisons une API traditionnelle telle que *scale-in* et *scale-out* pour déclencher une décision basée sur la stratégie que nous avons conçue, illustrée par (5) 1.1.

Conclusion

Le chapitre 8 conclut cette thèse en revisitant l'énoncé du problème, en résumant les contributions, en discutant des avantages, des idées et des limites de nos solutions proposées. Par la suite, nous discutons quelques-unes des orientations possibles et des idées qui pourraient créer de nouveaux défis à l'avenir sur la base de la contribution de cette thèse.

Bibliography

[AB99] Tarek F. Abdelzaher and Nina Bhatti. Web content adaptation to improve server overload behavior. *Comput. Networks*, 31(11-16):1563–1577, May 1999.

[AFZZ+14] Ahmed Amokrane, Mohamed Faten Zhani, Qi Zhang, Rami Langar, Raouf Boutaba, and Guy Pujolle. On satisfying green slas in distributed clouds. In *10th International Conference on Network and Service Management, CNSM Rio de Janeiro, Brazil, November 17-21*, pages 64–72, 2014.

[ALFZ+15] Ahmed Amokrane, Rami Langar, Mohamed Faten Zhani, Raouf Boutaba, and Guy Pujolle. Greenslater: On satisfying green slas in distributed clouds. *IEEE Trans. Network and Service Management*, 12(3):363–376, 2015.

[AMCGRS15] Gabriel A. Moreno, Javier Cámara, David Garlan, and Bradley R. Schmerl. Proactive self-adaptation under uncertainty: a probabilistic model checking approach. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Softwarengineering, ESEC/FSE , Bergamo, Italy*, pages 1–12, 2015.

[AMCGRS16] Gabriel A. Moreno, Javier Cámara, David Garlan, and Bradley R. Schmerl. Efficient decision-making under uncertainty for proactive self-adaptation. In *IEEE International Conference on Autonomic Computing, ICAC*, pages 147–156, 2016.

[APKSG13] Zahra Abbasi, Madhurima Pore, and Sandeep K. S. Gupta. Impact of workload and renewable prediction on the value of geographical workload management. In *Energy-Efficient Data Centers - Second International Workshop, E^2DC 2013, Berkeley, CA, USA, May 21, 2013.*, pages 1–15, 2013.

[APKSG14] Zahra Abbasi, Madhurima Pore, and Sandeep K. S. Gupta. Online server and workload management for joint optimization of electricity, cost and carbon footprint across data centers. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium, Phoenix, AZ, USA, May 19-23, 2014*, pages 317–326, 2014.

[ASB02] Tarek F. Abdelzaher, Kang G. Shin, and Nina Bhatti. Performance guarantees for web server end-systems: A control-theoretical approach. *IEEE Trans. Parallel Distrib. Syst.*, 13(1), January 2002.

[ASK14] Colin Atkinson, Thomas Schulze, and Sonja Klingert. Facilitating greener IT through green specifications. *IEEE Software*, 31(3):56–63, 2014.

[BAB12] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Gener. Comput. Syst.*, 28(5):14, May 2012.

[Bar05] L. Barroso. The price of performance. In *ACM Queue*, 2005.

[BJT+09] L. Bo, L. Jianxin, W. Tianyu, L. Qin, and Z. Liang. Enacloud: An energy-saving application live placement approach for cloud computing environments. In *Proc. of the IEEE Int. Conf. on Cloud Computing*, page 8. IEEE, 2009.

[BKT12] Christian Bunse, Sonja Klingert, and Schulze Thomas. Greenslas: Supporting energy-efficiency through contracts. In *Energy Efficient Data Centers - First International Workshop, E2DC*, pages 54–68, 2012.

[BR11] Michael Brown and Jose Renau. Rerack: power simulation for data centers with renewable energy generation. *SIGMETRICS Performance Evaluation Review*, 39(3):77–81, 2011.

[BSYV+09] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Comp. Syst.*, 25(6):599–616, 2009.

[CASS15] Mar Callau-Zori, Luciana Arantes, Julien Sopena, and Pierre Sens. Mercimiss: Should I turn off my servers? In *Distributed Applications and Interoperable Systems - 15th IFIP International Conference, DAIS Grenoble, France, June 2-4*, pages 16–29, 2015.

[CdCGdM+15] Paulo Rodrigo Cavalin, Maíra A. de C. Gatti, Tiago Gomes Pessoa de Moraes, Fabio Silva Oliveira, Claudio S. Pinhanez, Alexandre Rademaker, and Rogério Abreu de Paula. A scalable architecture for real-time analysis of microblogging data. *IBM Journal of Research and Development*, 59(2/3), 2015.

[CFF+11] Cinzia Cappiello, Alexandre Mello Ferreira, Maria Grazia Fugini, Pierluigi Plebani, and Monica Vitali. Business process co-design for energy-aware

adaptation. In *7th IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)*, pages 463–470, 2011.

[CGS+14] F. Chen, J. Grundy, J.G. Schneider, Y. Yang, and Q. He. Automated analysis of performance and energy consumption for cloud applications. In *ACM/SPEC Int Conf on Performance Engineering*, pages 514–521. ACM, 2014.

[CHT12] Changbing Chen, Bingsheng He, and Xueyan Tang. Green-aware workload scheduling in geographically distributed data centers. In *CloudCom*, pages 82–89. IEEE Computer Society, 2012.

[CIAP12] Houssem-Eddine Chihoub, Shadi Ibrahim, Gabriel Antoniu, and María Pérez. Harmony: Towards automated self-adaptive consistency in cloud storage. In *IEEE Int. Conf. on Cluster Computing*. IEEE, 2012.

[CJX14] Dazhao Cheng, Changjun Jiang, and Zhou Xiaobo. Heterogeneity-aware workload placement and migration in distributed sustainable datacenters. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium, Phoenix, AZ, USA, May 19-23, 2014*, pages 307–316, 2014.

[CKMP17] Stefania Costache, Samuel Kortas, Christine Morin, and Nikos Parlavantzas. Market-based autonomous resource and application management in private clouds. *Journal of Parallel and Distribured Computing*, 100:85–102, 2017.

[CPMK11] Stefania Costache, Nikos Parlavantzas, Christine Morin, and Samuel Kortas. An economic approach for application qos management in clouds. In *Euro-Par 2011: Parallel Processing Workshops - CCPI, CGWS, HeteroPar, HiBB, HPCVirt, HPPC, HPSS, MDGS, ProPer, Resilience, UCHPC, VHPC, Bordeaux, France, August 29 - September 2, 2011*, pages 426–435, 2011.

[CWC14] R.S. Chang C. Wu and H.Y. Chan. A green energy-efficient scheduling algorithm using the dvfs technique for cloud datacenters. *Future Generation Computer Systems*, 37:141–147, july 2014.

[DBADOL17] Simon Dupont, Salma Bouri, Frederico Alvares De Oliveira, and Thomas Ledoux. ElaScript: a DSL for Coding Elasticity in Cloud Computing. In *32nd ACM Symposium on Applied Computing - Track on Cloud Computing*, Proceedings of the 32nd ACM Symposium on Applied Computing - Track on Cloud Computing, Marrakesh, Morocco, 2017.

[DLAdOJT15] Simon Dupont, Jonathan Lejeune, Frederico Alvares de Oliveira Jr., and Ledoux Thomas. Experimental analysis on autonomic strategies for cloud

elasticity. In *2015 International Conference on Cloud and Autonomic Computing, Boston, MA, USA, September 21-25, 2015*, pages 81–92, 2015.

[DLJ+13] Wei Deng, Fangming Liu, Hai Jin, Chuan Wu, and Xue Liu. Multigreen: cost-minimizing multi-source datacenter power supply with online control. In *The Fourth International Conference on Future Energy Systems, e-Energy, Berkeley, CA, USA, May 22-24*, pages 149–160, 2013.

[DLJ+14] Wei Deng, Fangming Liu, Hai Jin, Bo Li, and Dan Li. Harnessing renewable energy in cloud datacenters: opportunities and challenges. *IEEE Network*, 28(1):48–55, 2014.

[DLJW13] Wei Deng, Fangming Liu, Hai Jin, and Chuan Wu. Smartdpss: Cost-minimizing multi-source power supply for datacenters with arbitrary demand. In *ICDCS*, pages 420–429. IEEE Computer Society, 2013.

[dOJL12] Frederico Alvares de Oliveira Jr. and Thomas Ledoux. Self-management of cloud applications and infrastructure for energy optimization. *SIGOPS Operating Systems Review*, 46(2):10–18, 2012.

[DSFH15] Corentin Dupont, Mehdi Sheikhalishahi, M. Facca Federico, and Fabien Hermenier. An energy aware application controller for optimizing renewable energy consumption in data centres. In *IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*, pages 195–204, Los Alamitos, CA, USA, 2015. IEEE Computer Society.

[EML+12] Maja Etinski, Margaret Martonosi, Kien Le, Ricardo Bianchini, and Thu D. Nguyen. Optimizing the use of request distribution and stored energy for cost reduction in multi-site internet services. In *Sustainable Internet and ICT for Sustainability, SustainIT, 4-5 October, Pisa, Italy*, pages 1–10, 2012.

[FFB13] D. Feitelson, E. Frachtenberg, and K. Beck. Development and deployment at facebook,. *IEEE Internet Computing*, 17:8–17, april 2013.

[FHB10] Daniel Fleder, Kartik Hosanagar, and Andreas Buja. Recommender systems and their effects on consumers: The fragmentation debate. In *Proc. of the 11th ACM Conf. on Electronic Commerce*, EC '10, pages 229–230, New York, NY, USA, 2010. ACM.

[FJLB15] Soodeh Farokhi, Pooyan Jamshidi, Drazen Lucanin, and Ivona Brandic. Performance-based vertical memory elasticity. In *IEEE Int. Conf. on Autonomic Computing*, pages 151–152, 2015.

[FYH+15] Weiwei Fang, Yuan Yao, Longbo Huang, Abhishek B. Sharma, Leana Golubchik, and Michael J. Neely. A comment on "power cost reduction in distributed data centers: A two time scale approach for delay tolerant workloads. *IEEE Trans. Parallel Distrib. Syst.*, 26(5):1495–1496, 2015.

[GBL+11] Íñigo Goiri, Ryan Beauchea, Kien Le, Thu D. Nguyen, Md. E. Haque, Jordi Guitart, Jordi Torres, and Ricardo Bianchini. Greenslot: Scheduling energy consumption in green datacenters. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011.

[GCWK12] Peter Xiang Gao, Andrew R. Curtis, Bernard Wong, and Srinivasan Keshav. It's not easy being green. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '12. ACM, 2012.

[GDK+14] Anshul Gandhi, Parijat Dube, Alexei Karve, Andrzej Kochut, and Li Zhang. Adaptive, model-driven autoscaling for cloud applications. In *11th International Conference on Autonomic Computing, ICAC, Philadelphia, PA, USA, June 18-20*, pages 57–64, 2014.

[GDXHJ14] Zehua Guo, Zhemin Duan, Yang Xu, and Chao H. Jonathan. Jet: Electricity cost-aware dynamic workload management in geographically electricity cost, electricity price, geographically distributed datacenters. *Computer Communications*, 50:162–174, 2014.

[GHBAK12] Anshul Gandhi, Mor Harchol-Balter, and Michael A. Kozuch. Are sleep states effective in data centers? *2012 International Green Computing Conference (IGCC)*, pages 1–10, 2012.

[GHMP08] Albert Greenberg, James Hamilton, David A. Maltz, and Parveen Patel. The cost of a cloud: Research problems in data center networks. *SIGCOMM Computer Communication Review*, (1):68–73, December 2008.

[GJNT06] Leonidas Georgiadis, Michael J. Neely, and Leandros Tassiulas. Resource allocation and cross-layer control in wireless networks. *Foundations and Trends in Networking*, 1(1), 2006.

[GKL+13] Íñigo Goiri, William Katsak, Kien Le, Thu D. Nguyen, and Ricardo Bianchini. Parasol and greenswitch: Managing datacenters powered by renewable energy. In *Proc. of the 8th Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13, pages 51–64. ACM, march 2013.

[GLN+12] Íñigo Goiri, K. Le, T. Nguyen, J. Guitart, J. Torres, and R. Bianchini. Greenhadoop: leveraging green energy in data-processing frameworks. In *ACM European Conference on Computer Systems, EuroSys*, pages 57–70, Apr 2012.

[GMR13] M. Ghamkhari and H. Mohsenian-Rad. Energy and performance management of green data centers: A profit maximization approach. *IEEE Transactions on Smart Grid*, 4(2):1017–1025, June 2013.

[GS13] Lawrence H. Goulder and Andrew Schein. Carbon taxes vs. cap and trade: A critical review. Climate Change Economics 19338, National Bureau of Economic Research, August 2013.

[GZG+16] Lin Gu, Deze Zeng, Song Guo, Yong Xiang, and Jiankun Hu. A general communication cost optimization framework for big data stream processing in geo-distributed data centers. *IEEE Trans. Computers*, 65(1):19–29, 2016.

[HAB07] Urs Hölzle and Luiz André Barroso. The case for energy-proportional computing. *Computer*, 40:33–37, 2007.

[HASX07] T. Horvath, T. Abdelzaher, K. Skadron, and Liu X. Dynamic voltage scaling in multitier web servers with end-to-end delay control. *IEEE Transactions on Computers*, 56:444–458, april 2007.

[HGG+14] Rui Han, Moustafa Ghanem, Li Guo, Yike Guo, and Michelle Osmond. Enabling cost-aware and adaptive elasticity of multi-tier cloud applications. *Future Generation Comp. Syst.*, 32:82–98, 2014.

[HGRN15] Md. E. Haque, Íñigo Goiri, Bianchini R., and Thu D. Nguyen. Greenpar: Scheduling parallel high performance applications in green datacenters. In *Proceedings of the 29th ACM on International Conference on Supercomputing, ICS'15, Newport Beach/Irvine, CA, USA, June 08 - 11, 2015*, pages 217–227, 2015.

[HH13] M. S. Hasan and E.N. Huh. Heuristic based energy-aware resource allocation by dynamic consolidation of virtual machines in cloud data center. *KSII Transactions on Internet and Information Systems*, 7(8), 2013.

[HKBB09] Kyong Hoon Kim, Anton Beloglazov, and Rajkumar Buyya. Power-aware provisioning of cloud resources for real-time services. In *Proceedings of the 7th International Workshop on Middleware for Grids, Clouds and e-Science, MCG, Champaign, IL, USA, November 30 - December 4*, page 1, 2009.

[HKCH16] Can Hankendi, Ayse Kivilcim Coskun, and Henry Hoffmann. Adapt&cap: Coordinating system- and application-level adaptation for power-constrained systems. *IEEE Design & Test*, 33(1):68–76, 2016.

[HLG+13] M. E. Haque, Kien Le, Iñigo Goiri, Ricardo Bianchini, and Thu D. Nguyen. Providing green slas in high performance computing clouds. In *International Green Computing Conference, IGCC, Arlington, VA, USA, June 27-29*, pages 1–11, 2013.

[HWY+10] Mi H, H. Wang, G. Yin, Y. Zhou, D. Shi, and L. Yuan. Online self-reconfiguration with performance guarantee for energy-efficient large-scale cloud computing data centers. In *IEEE International Conference on Services Computing*, pages 514–521. IEEE, 2010.

[IZM+16] Alexandru Iosup, Xiaoyun Zhu, Arif Merchant, Eva Kalyvianaki, Martina Maggio, Simon Spinner, Tarek F. Abdelzaher, Ole J. Mengshoel, and Sara Bouchenak. Self-awareness of cloud applications. *CoRR*, abs/1611.00323, 2016.

[KADOJDL14] Yousri Kouki, Frederico Alvares De Oliveira Jr., Simon Dupont, and Thomas Ledoux. A Language Support for Cloud Elasticity Management. In *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid*, pages 1–8, May 2014.

[KC03] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, Jan 2003.

[KC14] Pankaj Deep Kaur and Inderveer Chana. A resource elasticity framework for qos-aware execution of cloud applications. *Future Generation Comp. Syst.*, 37:14–25, 2014.

[KCH09] Evangelia Kalyvianaki, Themistoklis Charalambous, and Steven Hand. Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters. In *Proceedings of the 6th IEEE/ACM International conference on Autonomic Computing*, ICAC '09, pages 117–126, New York, NY, USA, 2009. ACM.

[KDNH15] E. Kern, M. Dick, S. Naumann, and T. Hiller. Impacts of software and its engineering on the carboon footprint of ict. *Environmental Impact Assessment Review*, 52:53–61, april 2015.

[KKJ11] Dara Kusic, Nagarajan Kandasamy, and Guofei Jiang. Combined power and performance management of virtualized computing environments serving session-based workloads. *IEEE Trans. Network and Service Management*, 8(3):245–258, 2011.

[KL12] Yousri Kouki and Thomas Ledoux. Csla: A language for improving cloud sla management. In *CLOSER*, pages 586–591. SciTePress, 2012.

[KL16] Fanxin Kong and Xue Liu. Greenplanning: Optimal energy source selection and capacity planning for green datacenters. In *7th ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS, Vienna, Austria, April 11-14,*, pages 5:1–5:10, 2016.

[KMAHR14] Cristian Klein, Martina Maggio, Karl-Erik Årzén, and Francisco Hernández-Rodriguez. Brownout: Building more robust cloud applications. In *Proc. of the 36th Int. Conf. on Software Engineering*. ACM, 2014.

[KPD+14] Cristian Klein, A. V. Papadopoulos, M. Dellkrantz, Martina Maggio, Karl-Erik Årzén, Francisco Hernández-Rodriguez, and E. Elmroth. Improving cloud service resilience using brownout-aware load-balancing. In *IEEE 33rd Int. Symposium on Reliable Distributed Systems*, pages 31–40, Oct 2014.

[KSB11] Sonja Klingert, Thomas Schulze, and Christian Bunse. Greenslas for the energy-efficient management of data centres. In *2nd International Conference on Energy-Efficient Computing and Networking*, pages 21–30, 2011.

[LCB+12] Zhenhua Liu, Yuan Chen, Cullen Bash, Adam Wierman, Daniel Gmach, Zhikui Wang, Manish Marwah, and Chris Hyser. Renewable and cooling aware workload management for sustainable data centers. In *SIGMETRICS*, pages 175–186. ACM, 2012.

[LFF12] G. Lami, F. Fabbrini, and M. Fusani. Software sustainability from a process-centric perspective. *Systems, Software and Services Process Improvement*, 52:97–108, april 2012.

[LLW+11] Zhenhua Liu, Minghong Lin, Adam Wierman, Steven H. Low, and Lachlan L. H. Andrew. Geographical load balancing with renewables. *SIGMETRICS Performance Evaluation Review*, 39(3):62–66, 2011.

[LOM15] Yunbo Li, Anne-Cécile Orgerie, and Jean-Marc Menaud. Opportunistic scheduling in clouds partially powered by green energy. In *IEEE International Conference on Data Science and Data Intensive Systems, DSDIS 2015, Sydney, Australia, December 11-13, 2015*, pages 448–455. IEEE Computer Society, 2015.

[LWL+14] Chao Li, Rui Wang, Tao Li, Depei Qian, and Yuan Jingling. Managing green datacenters powered by hybrid renewable energy systems. In *11th International Conference on Autonomic Computing, ICAC '14, Philadelphia, PA, USA, June 18-20.*, pages 261–272, 2014.

[MBL+13] Fabio Jorge Almeida Morais, Francisco Vilar Brasileiro, Raquel Vigolvino Lopes, Ricardo Araújo Santos, Wade Satterfield, and Leandro Rosa. Autoflex: Service agnostic auto-scaling framework for iaas deployment models. In *CCGRID*, pages 42–49. IEEE Computer Society, 2013.

[MG11] Peter Mell and Timothy Grance. The nist definition of cloud computing. Technical Report 800-145, National Institute of Standards and Technology (NIST), September 2011.

[MH12] Ming Mao and Marty Humphrey. A performance study on the vm startup time in the cloud. In *IEEE CLOUD*. IEEE, 2012.

[MHL+11] Simon Malkowski, Markus Hedwig, Jack Li, Calton Pu, and Dirk Neumann. Automated control for elastic n-tier workloads based on empirical modeling. In *Proceedings of the 8th International Conference on Autonomic Computing, ICAC, Karlsruhe, Germany, June 14-18, 2011*, pages 131–140, 2011.

[Mic14] Microsoft. Azure auto-scaling service, 2014. Available online at https://docs.microsoft.com/en-us/azure/cloud-services/cloud-services-how-to-scale.

[MTS+12] Jason Mars, Lingjia Tang, Kevin Skadron, Mary Lou Soffa, and Robert Hundt. Increasing utilization in modern warehouse-scale computers using bubble-up. *IEEE Micro*, 32(3):88–99, 2012.

[NDKJ11] S. Naumann, M. Dick, E. Kern, and T. Johann. The greensoft model: A reference model for green and sustainable software and its engineering. *Sustainable Computing: Informatics and Systems*, 1:294–304, april 2011.

[NH07] F. Nah and F. Hoon. A study on tolerable waiting time: How long are web users willing to wait? *Behavior and Information Technology*, 29(3):56–63, Feb 2007.

[NHL16] Zhaojie Niu, Bingsheng He, and Fangming Liu. Not all joules are equal: Towards energy-efficient and green-aware data processing frameworks. *2016 IEEE International Conference on Cloud Engineering (IC2E)*, pages 2–11, 2016.

[PDPBG10] Jeremy Philippe, Noel De Palma, Fabienne Boyer, and Olivier Gruber. Self-adaptation of service level in distributed systems. *Software Practice and Experience*, 40(3):259–283, March 2010.

[PP06] KyoungSoo Park and Vivek S. Pai. Comon: a mostly-scalable monitoring system for planetlab. In *SIGOPS Operating Systems Review*, volume 40, pages 65–74. ACM, 2006.

[PSZ+07] P. Padala, K.G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant. Adaptive control of virtualized resources in utility computing environments. In *Proc. of EuroSys*, pages 289–302. ACM, 2007.

[PWAJ15] Tharindu Patikirikorala, Liuping Wang, Colman Alan, and Han Jun. Differentiated performance management in virtualized environments using nonlinear control. *IEEE Trans. Network and Service Management*, 12(1):101–113, 2015.

[PWCH11] T. Patikirikorala, L. Wang, A. Colman, and J. Han. Hammerstein-wiener nonlinear model based predictive control for relative qos performance and resource management of software systems. In *Control Engineering Practice*, pages 49–61, 2011.

[RBW06] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006.

[RLXL10] Lei Rao, Xue Liu, Le Xie, and Wenyu Liu. Minimizing electricity cost: Optimization of distributed internet data centers in a multi-electricity-market environment. In *INFOCOM, 29th IEEE International Conference on Computer Communications, 15-19 March 2010, San Diego, CA, USA*, pages 1145–1153, 2010.

[RMOR13] J. Rogelj, D. L. McCollum, B.C. O'Neill, and K. Riahi. 2020 emissions levels required to limit warming to below 2 degree celcius. *Nature Climate Change*, 3(405–412), December 2013.

[RSRK07] S. Rivoire, M.A. Shah, P. Ranganathan, and C. Kozyrakis. Joulesort: A balanced energy-efficiency benchmark. In *ACM SIGMOD International Conference on Management of Data,*. ACM, 2007.

[RWUS12] Chuangang Ren, Di Wang, Bhuvan Urgaonkar, and Anand Sivasubramaniam. Carbon-aware energy capacity planning for datacenters. In *MASCOTS*, pages 391–400. IEEE Computer Society, 2012.

[SCJSC16] Prateek Sharma, Lucas Chaufournier, Prashant J. Shenoy, and Tay Y. C. Containers and virtual machines at scale: A comparative study. In *Proceedings of the 17th International Middleware Conference, Trento, Italy*, page 1, 2016.

[SDMD+16] Xiaoyu Shi, Jin Dong, Seddik M. Djouadi, Yong Feng, Xiao Ma, and Yefu Wang. Papmsc: Power-aware performance management approach for virtualized web servers via stochastic control. *J. Grid Comput.*, 14(1):171–191, 2016.

[Ser14] Amazon Web Service. Amazon cloudwatch, 2014. Available online at http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-cloudwatch.html.

[SN15] Andre Abrantes D. P. Souza and Marco A. S. Netto. Using application data for sla-aware auto-scaling in cloud environments. *CoRR*, abs/1506.05443, 2015.

[SS09] Christopher Stewart and Kai Shen. Some joules are more precious than others: Managing renewable energy in the datacenter. In *In HotPower*. ACM/USENIX, 2009.

[Sta14] Aikaterini Stamou. Systematic service level agreement (sla) data management, 10/03 2014. ID: unige:40738.

[TB15] Adel Nadjaran Toosi and Rajkumar Buyya. A fuzzy logic-based controller for cost and energy efficient load balancing in geo-distributed data centers. In *UCC*, pages 186–194. IEEE Computer Society, 2015.

[TL14] Marian Turowski and Alexander Lenk. Vertical scaling capability of openstack - survey of guest operating systems, hypervisors, and the cloud management platform. In *Service-Oriented Computing - ICSOC 2014 Workshops - WESOA; SeMaPS, RMSOC, KASA, ISC, FOR-MOVES, CCSA and Satellite Events, Paris, France*, pages 351–362, 2014.

[UUJNS11] Rahul Urgaonkar, Bhuvan Urgaonkar, Michael J. Neely, and Anand Sivasubramaniam. Optimal power cost management using stored energy in data centers. In *Proceedings of the ACM SIGMETRICS, International Conference on Measurement and Modeling of Computer Systems, San Jose,CA, USA, 07-11 June)*, pages 221–232, 2011.

[VAN08] Akshat Verma, Puneet Ahuja, and Anindya Neogi. pmapper: Power and migration cost aware application placement in virtualized systems. In *Proc. of the 9th ACM/IFIP/USENIX Int. Conf. on Middleware*, Middleware '08, pages 243–264, New York, NY, USA, 2008. Springer-Verlag.

[vLW10] Gregor von Laszewski and Lizhe Wang. *GreenIT Service Level Agreements*, pages 77–88. Springer US, 2010.

[VLWYH09] Gregor Von Laszewski, Lizhe Wang, Andrew J. Younge, and Xi He. Power-aware scheduling of virtual machines in dvfs-enabled clusters. In *CLUSTER*, pages 1–10. IEEE Computer Society, 2009.

[VRMCL09] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, 2009.

[WJSVSY07] Timothy Wood, Prashant J. Shenoy, Arun Venkataramani, and Mazin S. Yousif. Black-box and gray-box strategies for virtual machine migration. In *4th Symposium on Networked Systems Design and Implementation NSDI April 11-13, Cambridge, Massachusetts, USA*, 2007.

[WLDW10] Lizhe Wang, Gregor Von Laszewski, Jai Dayal, and Fugang Wang. Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with DVFS. In *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, CCGrid, 17-20 May, Melbourne, Victoria, Australia*, pages 368–377, 2010.

[WW11] Xiaorui Wang and Yefu Wang. Coordinating power control and performance management for virtualized server clusters. *IEEE Trans. Parallel Distrib. Syst.*, 22(2):245–259, 2011.

[WZL14] Huangxin Wang, Jean X. Zhang, and Fei Li. On time-sensitive revenue management and energy scheduling in green data centers. *CoRR*, abs/1404.4865, 2014.

[WZS05] Zhikui Wang, Xiaoyun Zhu, and Singhal Sharad. Utilization and slo-based control for dynamic sizing of resource partitions. In *Ambient Networks, 16th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM 2005, Barcelona, Spain*, pages 133–144, 2005.

[YF13] Lenar Yazdanov and Christof Fetzer. Vscaler: Autonomic virtual machine scaling. In *IEEE CLOUD*, pages 212–219. IEEE, 2013.

[YM13] Jumie Yuventi and Roshan Mehdizadeh. A critical analysis of power usage effectiveness and its use as data center energy sustainability metrics. *Energy and Buildings*, 64(WP131):90 – 94, 2013.

[ZWW11] Yanwei Zhang, Yefu Wang, and Xiaorui Wang. Greenware: Greening cloud-scale data centers to maximize the use of renewable energy. In *ACM/IFIP/USENIX 12th International Middleware Conference*, Lecture Notes in Computer Science, pages 143–164. Springer, 2011.

INSA de RENNES
Service des Formations

AVIS DU JURY SUR LA REPRODUCTION DE LA THESE SOUTENUE

Titre de la thèse:
Smart management of renewable energy in Clouds: from infrastructure to application

Nom Prénom de l'auteur : HASAN MD SABBIR

Membres du jury :
- Monsieur PAZAT Jean-Louis
- Madame MORIN Christine
- Monsieur LEDOUX Thomas
- Monsieur ROUVOY Romain
- Monsieur RUTTEN Eric
- Monsieur BOUVRY Pascal
- Monsieur PIERSON Jean-Marc

Président du jury : Christine Morin

Date de la soutenance : 03 Mai 2017

Reproduction de la these soutenue

Thèse pouvant être reproduite en l'état X
Thèse pouvant être reproduite après corrections suggérées

Fait à Rennes, le 03 Mai 2017

Signature du président de jury

Le Directeur,

M'hamed DRISSI

INSTITUT NATIONAL des SCIENCES APPLIQUÉES - RENNES

Résumé

Avec l'avènement des technologies de Cloud computing et son adoption, les entreprises et les institutions académiques transfèrent de plus en plus leurs calculs et leurs données vers le Cloud. Alors que ce progrès et ce modèle simple d'accès ont eu un impact considérable sur notre communauté scientifique et industrielle en termes de réduction de la complexité et augmentation des revenus, les centres de données consomment énormément d'énergie, ce qui se traduit par des émissions plus élevées de CO2. En réponse, de nombreux travaux de recherche se sont focalisés sur les enjeux du développement durable pour le Cloud à travers la réduction de la consommation d'énergie en concevant des stratégies d'efficacité énergétiques. Cependant, l'efficacité énergétique dans l'infrastructure du Cloud ne suffira pas à stimuler la réduction de l'empreinte carbone. Il est donc impératif d'envisager une utilisation intelligente de l'énergie verte à la fois au niveau de l'infrastructure et de l'application pour réduire davantage l'empreinte carbone.

Depuis peu, certains fournisseurs de Cloud computing alimentent leurs centres de données avec de l'énergie renouvelable. Les sources d'énergie renouvelable sont très intermittentes, ce qui crée plusieurs défis pour les gérer efficacement. Pour surmonter ces défis, nous étudions les options pour intégrer les différentes sources d'énergie renouvelable de manière réaliste et proposer un *Cloud energy broker* qui peut ajuster la disponibilité et la combinaison de prix pour acheter de l'énergie verte dynamiquement sur le marché de l'énergie et rendre les centres de données partiellement verts. Puis, nous introduisons le concept de la virtualisation de l'énergie verte, qui peut être vu comme une alternative au stockage d'énergie utilisé dans les centres de données pour éliminer le problème d'intermittence dans une certaine mesure. Avec l'adoption du concept de virtualisation, nous maximisons l'utilisation de l'énergie verte contrairement au stockage d'énergie qui induit des pertes d'énergie, tout en introduisant des Green SLA basé sur l'énergie verte pour le fournisseur de services et les utilisateurs finaux. En utilisant des traces réalistes et une simulation et une analyse approfondie, nous montrons que la proposition peut fournir un système efficace, robuste et rentable de gestion de l'énergie pour le centre de données.

Si une gestion efficace de l'énergie en présence d'énergie verte intermittente est nécessaire, la façon dont les applications Cloud modernes peuvent tirer profit de la présence ou l'absence d'énergie verte n'a pas été suffisamment étudiée. Contrairement aux applications Batch, les applications Interactive Cloud doivent toujours être accessibles et ne peuvent pas être programmées à l'avance pour correspondre au profil d'énergie verte. Par conséquent, cette thèse propose une solution *d'autoscaling* adaptée à l'énergie pour exploiter les caractéristiques internes des applications et créer une conscience d'énergie verte dans l'application, tout en respectant les propriétés traditionnelles de QoS. Pour cela, nous concevons un contrôleur d'application *green* qui profite de la disponibilité de l'énergie verte pour effectuer une adaptation opportuniste dans une application gérée par un contrôleur orienté performance. L'expérience est réalisée avec une application réelle sur Grid5000 et les résultats montrent une réduction significative de la consommation d'énergie par rapport à l'approche orientée performance, tout en respectant les attributs traditionnels de QoS.

Abstract

With the advent of cloud enabling technologies and adoption of cloud computing, enterprise and academic institutions are moving their IT workload to the cloud. Although this prolific advancement and easy to access model have greatly impacted our scientific and industrial community in terms of reducing complexity and increasing revenue, data centers are consuming enormous amount of energy, which translates into higher carbon emission. In response, varieties of research work have focused on environmental sustainability for Cloud Computing paradigm through energy consumption reduction by devising energy efficient strategies. However, energy efficiency in cloud infrastructure alone is not going to be enough to boost carbon footprint reduction. Therefore, it is imperative to envision of smartly using green energy at infrastructure and application level for further reduction of carbon footprint.

In recent years, some cloud providers are powering their data centers with renewable energy. The characteristics of renewable energy sources are highly intermittent which creates several challenges to manage them efficiently. To overcome the problem, we investigate the options and challenges to integrate different renewable energy sources in a realistic way and propose a Cloud energy broker, which can adjust the availability and price combination to buy Green energy dynamically from the energy market in advance to make a data center partially green. Later, we introduce the concept of Virtualization of Green Energy, which can be seen as an alternative to energy storage used in data center to eliminate the intermittency problem to some extent. With the adoption of virtualization concept, we maximize the usage of green energy contrary to energy storage which induces energy losses, while introduce Green Service Level Agreement based on green energy for service provider and end users. By using realistic traces and extensive simulation and analysis, we show that, the proposal can provide an efficient, robust and cost-effective energy management scheme for data center.

While an efficient energy management in the presence of intermittent green energy is necessary, how modern Cloud applications can take advantage of the presence/absence of green energy has not been studied with requisite effort. Unlike Batch applications, Interactive Cloud applications have to be always accessible and can not be scheduled in advance to match with green energy profile. Therefore, this thesis proposes an energy adaptive autoscaling solution to exploit applications internal to create green energy awareness in the application, while respecting traditional QoS properties. To elaborate, we design green energy aware application controller that takes advantage of green energy availability to perform opportunistic adaptation in an application along with performance aware application controller. Experiment is performed with real life application at Grid5000 and results show significant reduction of energy consumption while respecting traditional QoS attributes compared to performance aware approach.

INSA RENNES

N° d'ordre : **17ISAR 11 / D17 -11**

Institut National des Sciences Appliquées de Rennes
20, Avenue des Buttes de Coësmes - CS 14315 - F-35043 Rennes Cedex
Tél : 02 23 23 82 00 – Fax : 02 23 23 83 96