Πανεπιστήμιο Θεσσαλίας

Τμήμα Ηλεκτρολόγων Μηχανικών και
Μηχανικών Ηλεκτρονικών Υπολογιστών

Μεταπτυχιακό Πρόγραμμα:
Επιστήμη των Υπολογιστών, Τηλεπικοινωνιών και
Δικτύων

Προβλήματα Ανάθεσης Πόρων σε Δίκτυα Νεφελώδους Υπολογισμού

Χατζόπουλος Δημήτρης

Επιβλέποντες Καθηγητές:
Κουτσόπουλος Ιορδάνης
Τασιούλας Λέανδρος
Κατσαρός Δημήτριος

ΒΟΛΟΣ, 2013

Περίληψη

Η νεφελώδης υπολογιστική (cloud computing) είναι ένα νέο αναβαθμισμένο μοντέλο της τεχνολογίας των πληροφοριών που εμφανίστηκε τα τελευταία χρόνια και έχει ήδη αλλάξει τον τεχνολογικό κόσμο. Προσφέρει ένα σύνολο υπηρεσιών μέσω του διαδικτύου χρησιμοποιώντας μια σειρά απο κοινόχρηστους πόρους. Το μοντέλο τιμολόγησης των υπηρεσιών που παρέχει η νεφελώδης υπολογιστική ακολουθεί ένα σύστημα χρέωσης μόνο των πόρων που χρησιμοποιήθηκαν απο τον πελάτη. Οι υπηρεσίες της νεφελώδους υπολογιστικής παρέχονται μέσω υπολογιστικών συστήματων τα οποία ονομάζονται κέντρα δεδομένων (data centers).

Δυστυχώς, τα κέντρα δεδομένων απαιτούν μεγάλα πόσα ενέργειας. Μερικές εγκαταστάσεις χρειάζονται ενέργεια εκατό φορές παραπάνω από ένα τυπικό κτιριακό συγκρότημα. Ένας φιλικός προς το περιβάλλον και οικονομικά αποτελεσματικός τρόπος για να περιοριστεί το λειτουργικό κόστος ενός κέντρου δεδομένων είναι η εγκατάσταση ανανεώσιμων πηγών ενέργειας. Ωστόσο, η παραγωγή ηλεκτρικής ενέργειας απο τις ανανεώσιμες πηγές ενέργειας δεν είναι ντετερμινιστική. Προκειμένου να είναι εγγυημένη η σταθερή τροφοδοσία ρεύματος για το κέντρο δεδομένων, χρησιμοποιείται επίσης ενέργεια από συμβατικό δίκτυο ηλεκτρικής ενέργειας. Αξίζει να σημειωθεί πως η τιμή μίας μονάδας ηλεκτρικής ενέργειας που προέρχεται από ανανεώσιμη πηγή ενέργειας είναι σχεδόν μηδέν. Επίσης η τιμή μίας μονάδας που προέρχεται από ένα συμβατικό δίκτυο ηλεκτρικής ενέργειας διαφέρει απο τοποθεσία σε τοποθεσία και είναι χρονικά μεταβαλλόμενη.

Η διπλωματική διατριβή που κρατάτε στα χέρια σας διερευνά το πρόβλημα της εξυπηρέτησης αιτήσεων για υπηρεσίες νεφελώδους υπολογιστικής σε ένα δίκτυο απο κέντρα δεδομένων που βρίσκονται σε διαφορετικές γεωγραφικές περιοχές. Μια αίτηση εξυπηρετείται μεσω μίας εικονικής μηχανής (virtual machine). Κάθε κέντρο δεδομένων είναι συνδεδεμένο σε ένα συμβατικό δίκτυο ηλεκτρικής ενέργειας και επιπλέον υποστηρίζεται από μία ανανεώσιμη πηγή ενέργειας.

Το πρόβλημα αναδεικνύεται μέσα από ένα σύστημα αποτελεσματικής χρήσης της ενέργειας μεταξύ ενός χρονικά μεταβαλλόμενου ανά μονάδα ηλεκτρικής ενέργειας κόστους από τον συμβατικό πάροχο ηλεκτρικής ενέργειας και τον απρόβλεπτο και χρονικά μεταβαλλόμενο ενεργειακό εφοδιασμό απο μία ανανεώσιμη πηγή ενέργειας. Ο αντικειμενικός σκοπός είναι να ελαχιστοποιηθεί το ολικό λειτουργικό κόστος κατανάλωσης ενέργειας όσον αφορά το διαχειριστή (cloud provider) και να μειωθούν οι περιβαλλοντικές επιπτώσεις των επιμέρους διαδικασιών. Για να επιτευχθεί αυτός ο στόχος θεωρούμε πως κάθε αίτημα αρχικά λαμβάνεται και τοποθετείται σε μία κεντρική ουρά. Κάθε αίτημα συνδέεται με ένα αίτημα εικονικής μηχανής που έχει κάποιες ανάγκες από πόρους αλλά και μία χρονική προθεσμία πριν απο την οποία πρέπει να ικανοποιηθεί. Ο διαχειριστής του δικτύου των κέντρων δεδομένων θα δημιουργήσει μία εικονική μηχανή με τις ανάλογες απαιτήσεις πόρων και θα την εκτελέσει πριν την εκπνοή της χρονικής προθεσμίας. Η εκτέλεση της αίτησης εγγυάται απο ένα συμβόλαιο (service level agreement) που γίνεται ανάμεσα στον αιτούμενο για την υπηρεσία και τον πάροχο της υπηρεσίας.

Προτείνεται ένας αλγόριθμος που παίρνει σαν δεδομένα το κόστος ανά μονάδα ρεύματος απο τον πάροχο του και την ικανότητα παραγωγής ενέργειας απο την ανανεώσιμη πηγή ενέργειας για ένα χρονικό ορίζοντα και αποφασίζει για κάθε αίτηση. Ο προαναφερθέν αλγόριθμος συγκρίνεται με έναν '' άπληστο'' αλγόριθμο και με έναν '' απλοϊκό ''.

Επείσης, μετατρέπουμε το πρόβλημα δημιουργίας εικονικών μηχανών και ανάθεσης τους στα κατάλληλα κέντρα δεδομένων για την εξυπηρέτηση

των αιτήσεων για μια υπηρεσία νεφελώδους υπολογιστικής σε ένα πρόβλημα εύρεσης συντομότερου μονοπατιού σε ένα γράφο. Επιπλέον, παραθέτουμε έναν απλό μηχανισμό τιμολόγησης κάθε αιτήματος. Τέλος, χρησιμοποιούμε πραγματικά δεδομένα αιτήσεων σε συστήματα νεφελώδους υπολογιστικής και παραγωγής ενέργειας απο ανανεώσιμες πηγες σαν είσοδο στα αριθμητικά πειράματα μας.

University of Thessaly

Department of Electrical and Computer Engineering

Master Program:
Science of Computer and Communication Engineering

Resource allocation problems in cloud computing networks

Author:
Dimitrios Chatzopoulos

Supervisors:
Iordanis Koutsopoulos
Leandros Tassiulas
Dimitrios Katsaros

VOLOS 2013

Part of this work presented in the IEEE international conference on Communications (ICC) 2013 with title :
Dynamic Virtual Machine Allocation in Cloud Server Facility Systems with Renewable Energy Sources [28]
and co-authors:
Iordanis Koutsopoulos, George Koutitas and Ward Van Heddeghem.

Abstract

Cloud computing is a new upgraded information technology (IT) model which came up the last few years and has already changed the technological world. It offers a set of services (software, application, network, storage, infrastracture) over the internet by using a set of shared resources. The pricing model of cloud computing services follows the pay-as-you-go model which means that customers pay only for the resources they use. Cloud computing services are hosted in computer systems which are called data centers.

Unfortunately, data centers require large amounts of energy. For example, some facilities have power densities more than 100 times that of a typical office building. An environmental friendly and cost efficient way to reduce the operational costs of a data center is the installation of renewable energy sources (RESs). However, the power generation of RESs is not deterministic and time-variant. In order to be guaranteed a stable power supply for the data center, power from the power grid also used. It is worth mentioning that the price for a unit of power is almost zero for a RES but not for the power grid which price also changes by time.

This thesis explores the problem of virtual machine (VM) allocation over a network of cloud server facilities which are deployed in different geographical areas. Each cloud server facility is connected to the conventional power grid network and in addition supported by an attached renewable energy source. We address the problem of energy efficient task allocation in the system in the presence of a time varying grid energy price and the unpredictability and time variation of provisioned power by the RES. The objective is to reduce the total cost of power consumption associated to the operator and to reduce the environmental impact related to the processing of the tasks.

To achieve this goal we consider that all third-party VM requests initially arrive in a central queue.Each task request is associated with a VM request with some resource requirements and a deadline by which it needs to be completed. The cloud provider has to create a VM with the resource requirements specified at the request and execute the VM before the deadline. The task execution is quaranted by a service level agreement (SLA) between the client (i.e the task owner) and the cloud provider.

We propose an online algorithm with given lookahead horizon, in which the grid power prices and patterns of output power of the RESs are known a priori and we compare it with a greedy online algorithm and a naive algorithm. Also we transform the problem into a shortest-path problem under the assumption that all the stochastic parameters and the requests are known for a specific time horizon. Finally, we

propose a simple pricing mechanism who determines the price of the request execution.

Numerical results on real traces of cloud traffic and renewable source generation patterns are encouraging in terms of the performance of our techniques and motivate further research on the topic.

Contents

List of Tables

List of Figures

1 Cloud Computing

1.1 Cloud Computing Basics

Cloud computing[4][17] is a way to provide services over the network to remote customers. It might be deployed in an private, public or hybrid manner and has the following essential characteristics: customers buy as many resources as they want providing the illusion of infinite offer, whenever and for as long as they need to. As a result, up and front hardware/software commitment is eliminated and the ability to pay for use of computing resources on a short-term basis is granted. Resources can be accessed through the network from any computer, providing also geographical elasticity.

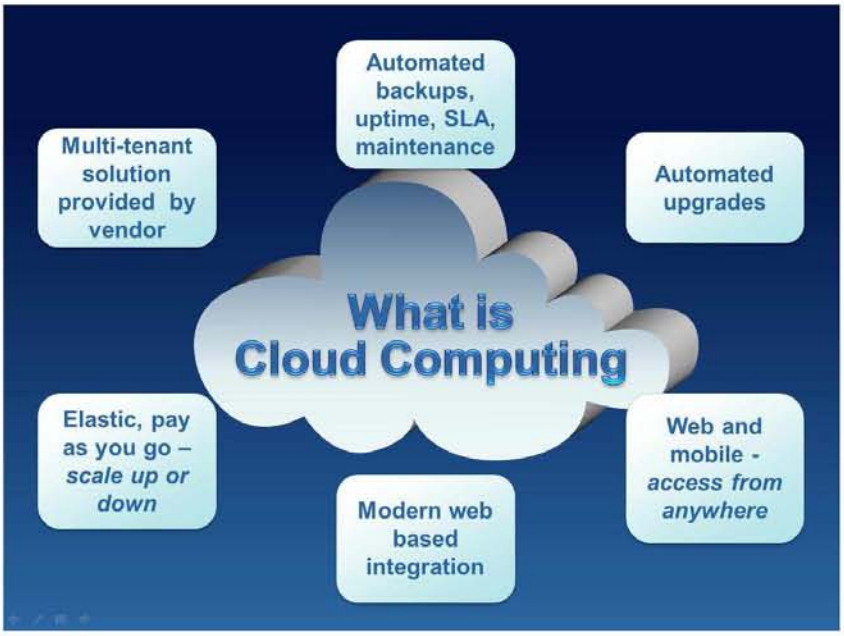

Figure 1: What is Cloud Computing.

The most popular types of these services are software (SaaS), platform (PaaS), network (NaaS) or hardware (IaaS) oriented. From the client's point of view, resources are utilized almost utterly because of the pay-as-you-go model reducing the cost of managing hardware and software. Furthermore, the fault tolerance problem does not exist any more (real hardware infrastructure is transparent). From the application's point of view there is no actual difference, although in reality applications run in virtual machines. Virtual machines are running on the same physical machine reacting like applications in conventional systems.

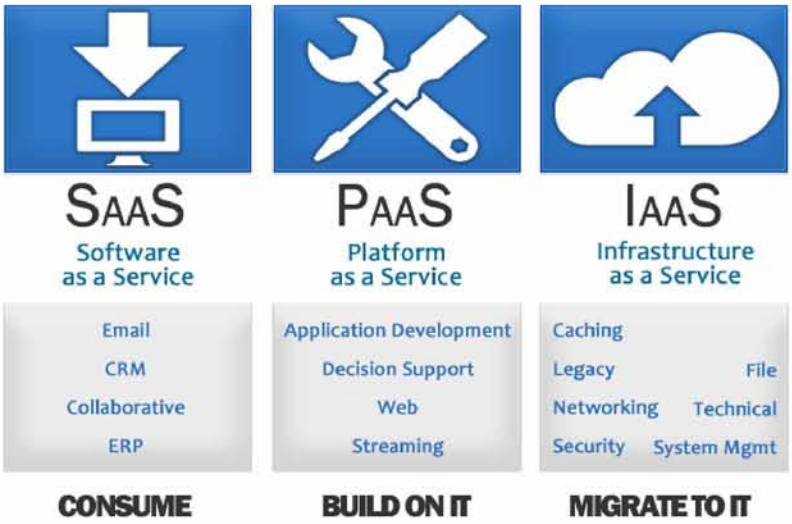

Figure 2: Types of cloud computing services.

1.2 Pay-as-you-go model

The pay-as-you-go model offers the ability to cloud customers to pay only for the resourses they use. While in the case of not using a cloud service, the customer has to buy the maximum required resources he needs and as a result pay more money.

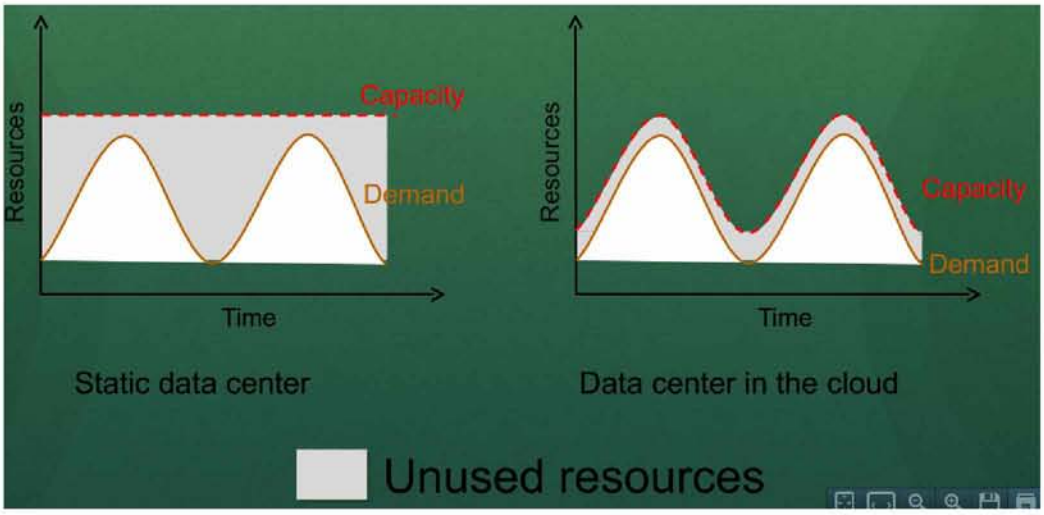

Figure 3: Pay by use instead of provisioning for peak [26].

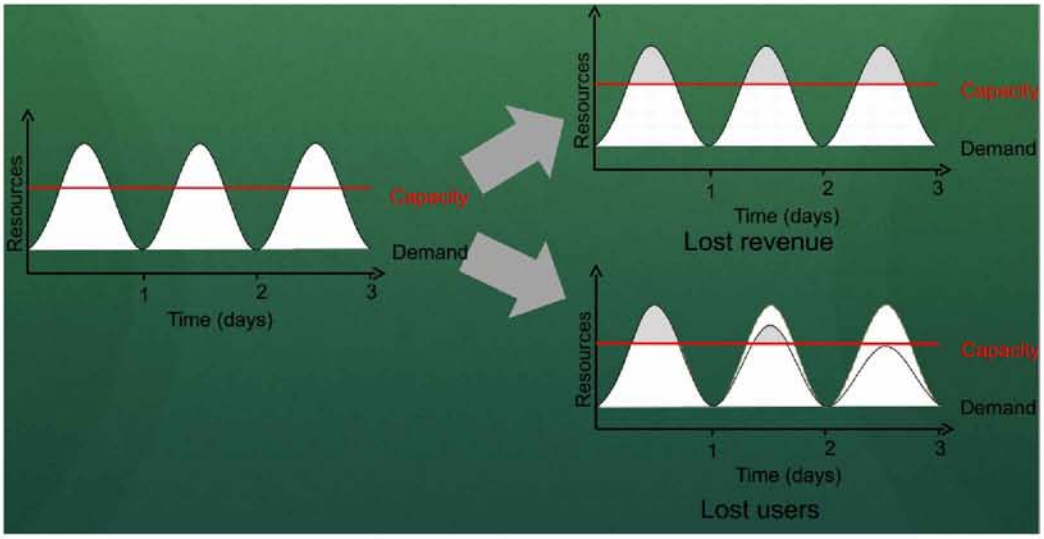

Figure 4: Penalty for under-provisioning [26].

1.3 Virtualization

The basic concept that differentiates cloud computing from software oriented architectures (SOA) is virtualization[16] and more specifically virtual machines[15]. Virtualization is the creation of a virtual version of something, such as a hardware platform, operating system, a storage device or network resources. A virtual machine (VM) is a software implementation of a machine that executes programs like a physical machine. A system virtual machine provides a complete system platform which supports the execution of a complete operating system.

An essential characteristic of a virtual machine is that the software running inside is limited to the resources and abstractions provided by the virtual machine it cannot break out of its virtual world. Classic benefits of virtualization include improved utilization, manageability, and reliability of systems. Benefits of virtualization have great appeal across a broad range of both server and client systems, more specifically in isolation, consolidation and migration. Isolation can improve overall system security and reliability, consolidation separates individual workloads onto a single physical platform and migration makes it possible to decouple the guest from the hardware on which it is running.

Two basic concepts in the cloud are the emulator and the hypervisor. An emulator is hardware or software or both that duplicates (or emulates) the functions of a first computer system in a different second computer system, so that the behavior of the second system closely resembles the behavior of

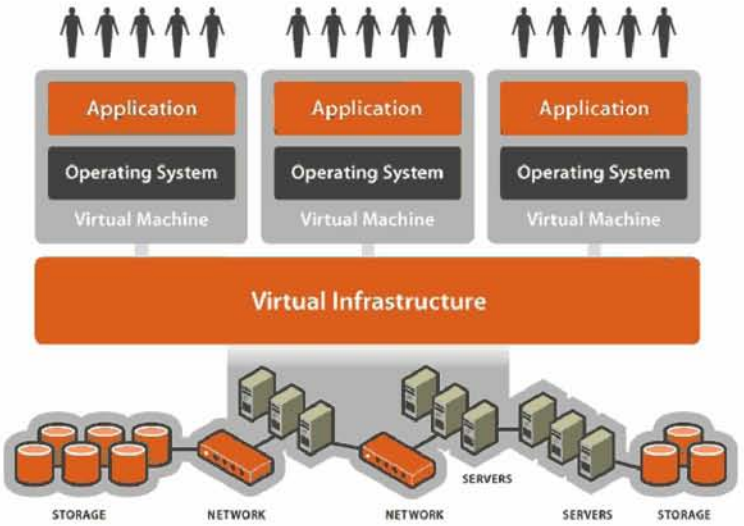

Figure 5: Example of virtualization.

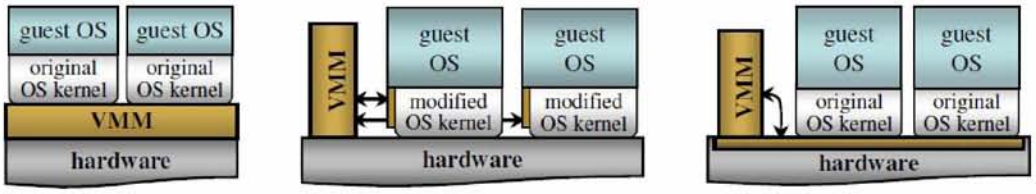

Figure 6: types of virtualization

the first system and a hypervisor (also called a virtual machine manager) is a program that allows multiple operating systems to share a single hardware host and controls and allocates host's resources to virtual machines.

As mentioned before, virtualization offers the ability of migrating a virtual machine from a physical machine to another, applications themselves and their corresponding processes are not aware that a migration is occurring because hypervisors allow migrating an OS as it continues to run. This is termed hot or live migration[25] and opposed to cold or pure stop-and-copy migration in which virtual machine halts, all memory pages copied to the destination physical machine and then virtual machine continues executing. Although live migration can take more time because some memory pages could be sent more than once, it offers a high probability of zero virtual machine downtime. Migration increases the system performance because it offers the ability of load balancing and more general efficient resource allocation technics and it helps cloud provider to manage his system (ie shut down/update/upgrade/repair some physical machines without stopping

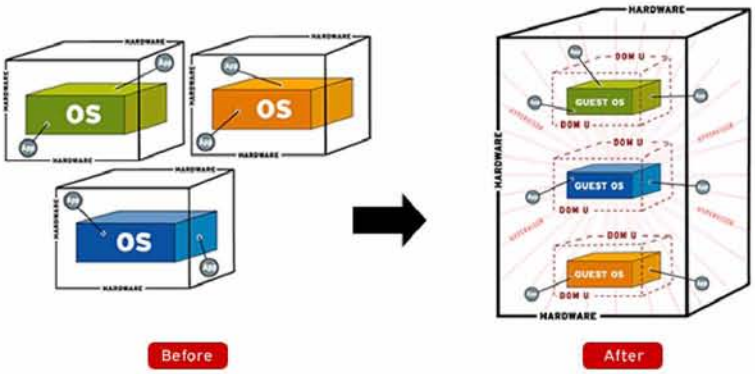

Figure 7: Virtualization: before and after.

providing the service). Unfortunately, sometimes a virtual machine must be halted to complete the migration. The time the virtual machine is halted called downtime and this is the migration cost in the case of live migration. On the other hand, in the case of cold migration, migration cost equals to the time needed to send all the useful virtual machine's memory pages plus the time needed to halt and restart on the new physical machine, which is ideally almost zero.

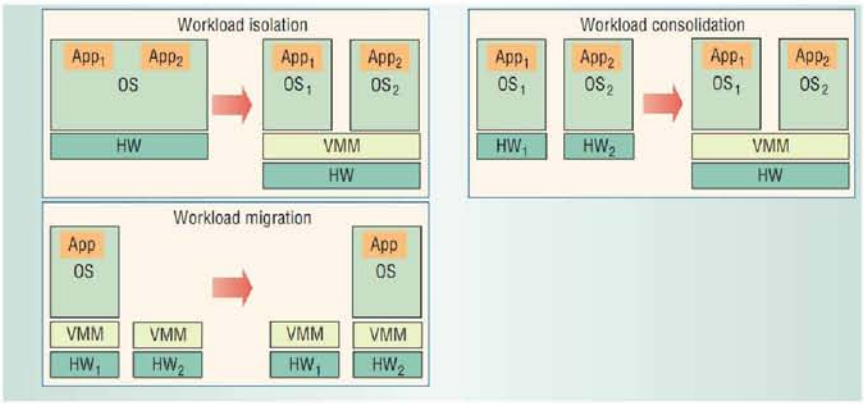

Figure 8: Benefits of virtualization.

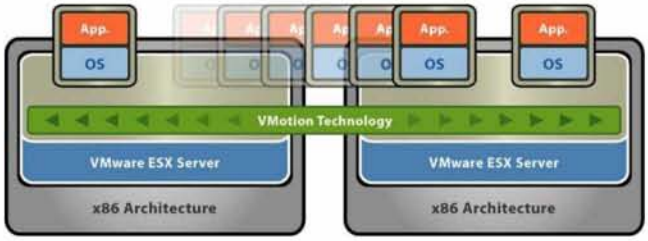

Figure 9: Virtual machine migration.

1.4 Cloud Providers

The cloud provider is a service provider that offers customers services available via a private (private cloud) or public network (cloud). Usually, he exposes the required resources for the service via the Internet. Table 1 contains some popular cloud service providers for the four most popular categories of cloud services.

IAAS	PAAS	SAAS	STORAGE
Amazon	Amazon	Abiquo	3X
AT&T	Appistry	AccelOps	Amazon
BlueLock	AppScale	Akamai	Asigra
CA	CA	AppDynamics	Axcient
Cloudscaling	Engine Yard	Apprenda	Carbonite
Datapipe	FlexiScale	CloudOptix	Caringo
ENKI	Force.com	Cloud9	Cleversafe
Enomaly	gCloud3	CloudSwitch	Cloud Attached Storage
Eucalyptus	GigaSpaces	CloudTran	Doyenz
GoGrid	Gizmox	Cumulux	eFolder
HP	Google	Eloqua	EVault
Joyent	GridGain	FinancialForce	Intronis
Layered Tech	LongJump	Intacct	Mezeo
Logicworks	Microsoft	Marketo	Nasuni
NaviSite	OpenStack	NetSuite	Nirvanix
OpSource	OrangeScape	Oracle	Scality
Rackspace	OS33	Pardot	StorSimple
Savvis	OutSystems	Salesforce.com	SugarSync
Terremark	RightScale	SAP	Vembu
Verizon	ThinkGrid		

Table 1: Popular cloud service providers [5]

Figure 10: A Google data center outside Atlanta, Ga.

2 Data Centers

2.1 Data Center Basics

A data center is a facility used to house computer systems and associated components, such as network and storage systems. Data centers may be private, where the entire facility is devoted to hosting applications belonging to the facility owner, or shared, where the facility owner leases out portions of the facility to different application providers. A shared data center may be operated as one of three kinds of services, depending on the interface exposed to the lessees: Infrastructure-as-a-Service (IaaS) exposes the lowest-level interface, and is essentially the leasing out of physical servers; Platform-as-a-Service (PaaS) is one level higher, and leases out virtual CPUs and disks; and finally, Software-as-a-Service (SaaS) leases out hosted software. Data centers generally include redundant or backup power supplies, redundant data communications connections, environmental controls and security devices. Large data centers are industrial scale operations using as much electricity as a small town.

Figure 11: A central cooling plant in Googles Douglas County, Georgia, data center.

2.2 Data Center Energy Consumption

Ideally, a data center should consume only as much energy as is needed to process incoming requests. Every request require some amount of energy to execute. In reality processing the request would additionally incur a number of energy overheads: energy used by idling resources, by air conditioners that cool the servers processing the request, and energy wasted in inefficient power delivery to the servers, among other overheads. The amount of these overheads depends on the energy (in)efficiency of the data center. An ideal data center would minimize these overheads. We enumerate two target properties of a data center that capture this idea:

1. **Power-proportionality:** This property states that executing a given job consumes a minimal amount of compute energy, irrespective of how much time it takes energy consumed by IT resources being proportional to work done. This is possible only if base-line power consumption[1] is zero. In other words, idle resource power consumption must be zero.

[1]power consumed when no job is being executed

2. **Power Utilization Efficiency (PUE) Close To 1:** This property states that the energy consumed during job execution is within a small margin of the amount of useful work done. In other words, the constant of proportionality relating energy consumed to useful work done should be close to 1. PUE is defined as the total energy consumed by the data center, divided by the total energy consumed by the servers in the data center. As energy consumed by a data center is divided between servers (useful work) and the power, cooling, and networking infrastructure that supports the correct functioning of servers (not useful work), PUE can be much larger than 1.

Together, these properties assert that the power consumed by a data center is a minimal function of its load. Data center power proportionality requires power consumption to track load, and eliminates overheads from idle resource power consumption. Low data center PUE ties power consumption closely to useful work done and minimizes overheads from support equipment power consumption. Thus, a data center that is power proportional and has a PUE of 1 would consume only as much energy as the application logic requires from the IT equipment.

The figure above compares an ideal power consumption curve, as described in the previous section, with the prevalent reality. The differences between these curves signal the presence of various inefficiencies in current data center design and operation. We identify two problem areas:

1. Energy consumption by idle resources
2. Energy consumption by support equipment

2.2.1 Energy consumption by idle resources

The ideal data center consumes zero energy under zero load. The reality, however, is that in inadequately managed facilities, servers consume almost as much energy when idle or lightly loaded, as when heavily loaded. This is the reason for the high offset in the power vs load curve of the average data center (figure 12).

Many server components have the ability to operate in multiple power modes, so that they can be manipulated to consume power proportional to their load, or desired level of performance. However, there are several challenges to this approach:

- Performance Tradeoff: Switching between power modes takes time and can translate to degraded performance if load goes up unexpectedly. Most services can tolerate very little, if any, performance degradation.

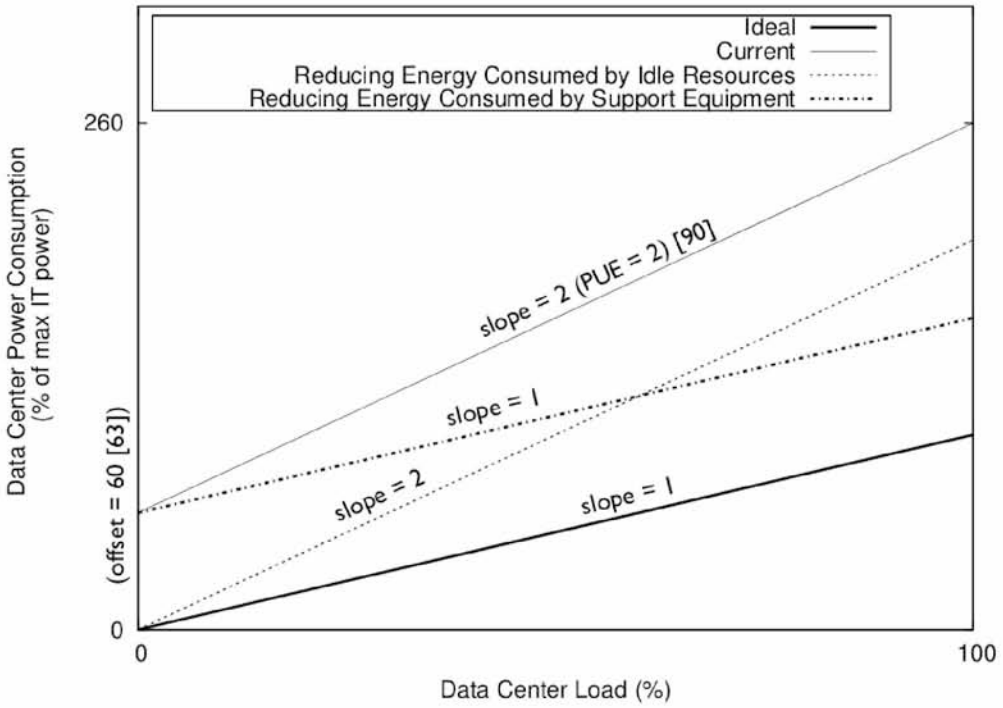

Figure 12: Schematic Diagram Of Data Center Power Consumption As A Function Of Load [27]

- Load Unpredictability: The load on a given server can be impacted by a plethora of factors, including time of day, day of year, current world affairs, geography, and flash crowds, among others, making it very hard to predict accurately. Power managing servers without adequate foreknowledge of their anticipated load can lead to significant performance degradation.

- Short Idle Times: Load spread can also vary continually, leading to short idle times for most servers. This means that the time and energy cost of switching them to lower power modes is often not worth the potential energy saving from the switch.

There are two basic approaches on minimizing energy consumption by idle resources in a data center. The first approach tries to reduce power consumption by idle resources by finding ways to enable switching them to lower power modes (or turning them off). This approach relies on designing mechanisms that improve the predictability and length of resource idle periods, to enable effective power-down. The second approach tries to eliminate (or

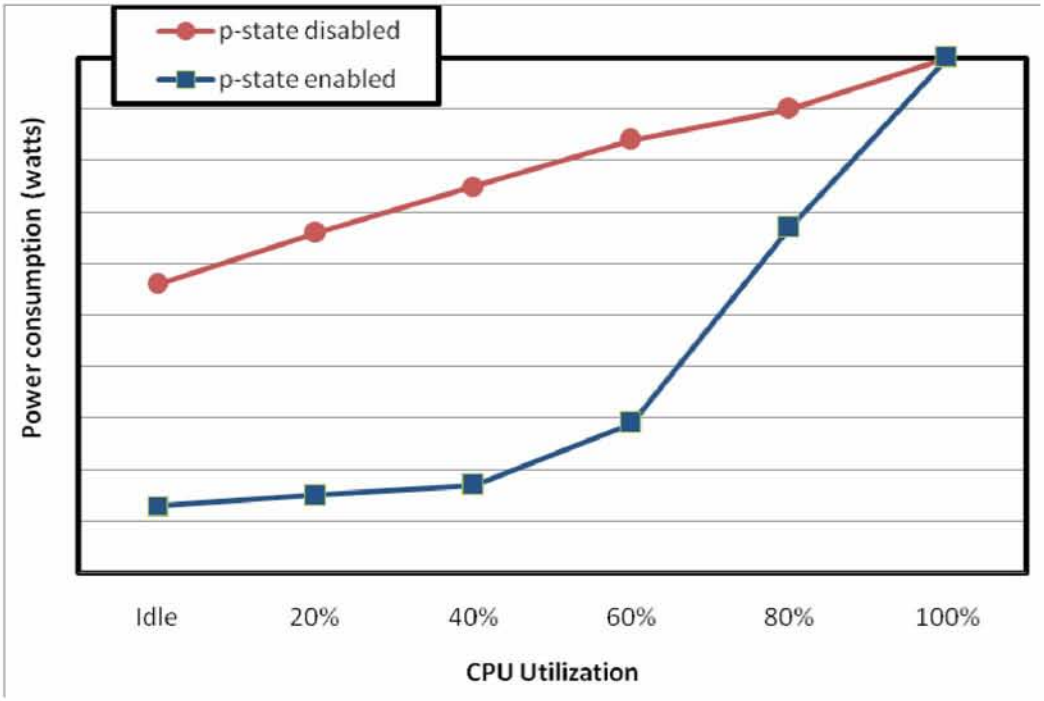

Figure 13: Impact of p-state on Power Consumption [24]

reduce) the presence of idle resources at all, by provisioning fewer resources, and over-subscribing them.

Examples of the first approach include disk power management solutions. Dynamic voltage and frequency scaling (DVFS) is a mechanism that allows CPU power to be manipulated to match its utilization. This is a useful tool, but needs an effective management framework that can maximize its benefit, by enabling sufficiently long idle CPU periods.

In recent years, x86 server processors have begun to incorporate the power saving architectures that have been common in both desktop and laptop computers. Enabling this feature can result in overall system power savings of up to 20%. The power saving is achieved by reducing the frequency multiplier (Frequency identier or FID) and the voltage (Voltage identier or VID) of the CPU. The combination of a specic CPU frequency and voltage is known as a performance state (p-state). Altering the p-state can reduce a servers power consumption when at low utilization but can still provide the same peak level of performance when required. The switch between p-states is dynamically controlled by the operating system and occurs in micro-seconds, causing no perceptible performance degradation

Resource over-subscription solutions typically employ a power-tracking

and capping approach. Power-tracking, as the name implies, is a mechanism to monitor power use, while power-capping prevents resources from exceeding a given (tunable) power cap. These are essentially safety mechanisms to enable resource oversubscription without the danger of overload and its repercussions.

2.2.2 Energy consumption by support infrastructure

In addition to the servers and IT equipment that are doing directly useful work, data centers contain a considerable amount of support infrastructure like power distribution and cooling equipment, that enables the IT equipment to function correctly, but does not contribute directly to useful work done. In the ideal data center, the energy consumption of the support equipment should be a small fraction of the energy consumption of the IT equipment. In reality, however, support equipment consumes a comparable amount of energy to the IT equipment. This leads to the steep slope of the power vs load curve of the average data center (figure 12). Support infrastructure performs the following functions:

- Cooling: Traditional data center cooling infrastructure consists of a chiller unit to chill the coolant used (water or air) and fans to direct cool air towards the servers and hot air away from the servers. These are both intrinsically power-hungry processes.

- Power Delivery: Power is typically delivered to a data center as high voltage AC power. After that it stepped down to lower voltage AC power for distribution to racks for use by servers and other IT equipment. Inside this IT equipment, power supplies convert the AC power to the DC power needed for digital electronics. For every Watt of energy used to power servers, up to 0.9 W can be lost through this series of power conversions. Additional power is needed to cool the conversion equipment.

- Power Backup: In order to prevent outages, data centers use a backup power supply that can kick in temporarily if the primary supply fails. Traditionally, this backup takes the form of a central UPS. Power to the facility flows through the UPS, charging it, and is then routed to the racks. Significant power loss can result from this model, as the average UPS has an efficiency of only about 92%.

Solutions have been proposed to address each of the power overheads from support equipment. A highly effective solution to reduce cooling power

overheads is free cooling, a system that uses ambient air for facility cooling, thus obviating the need for power-hungry chillers. It has been shown that free cooling can help bring data center PUE down to as low as 1.07 . However, a severe limiting factor for this solution is the requirement that ambient temperatures be suitable for use in facility cooling which does not hold for a majority of extant data centers. Power delivery efficiency has been shown to improve significantly by supplying the data center with DC power instead of AC power. However, this shift also comes at a significant deployment cost. Finally, it has been demonstrated that moving from a central UPS power backup solution to a distributed model with each server backed up by its own battery can eliminate the power loss through UPS inefficiencies. Finally, another approach to reducing support infrastructure energy consumption is to power them down when not needed.

3 Renewable Energy Sources (RES)

3.1 Types of renewables used in data centers

There are many forms of on-site alternative power generation, but some are more applicable to critical facilities than others. Here is a look at the pros and cons of the most practical alternatives for a critical facility such as a data center.

3.1.1 Solar Power

Currently, photovoltaic (PV) panels are the most commonly used source of alternative power for buildings. However, they require a lot of roof or facade space in order to provide enough power to supply even a portion of the full demand of a data center.Moreover, solar power can only be tapped during daylight hours. While it is possible to store solar energy in valve-regulated lead-acid or wet-cell batteries for nighttime use, this is rarely practical. At night, most facilities will still need to rely on the grid and also will need to maintain a diesel-powered emergency generator. Most data center owners that have PV systems use them to reduce a facilitys energy load and operating cost. For example, PVs can be employed for peak shaving when the demand and associated cost of electricity is the highest, during peak daylight hours. In addition, if the data center generates more power than it consumes during the day, the excess energy can be sold back to the grid to reduce electricity bills. Also on the plus side, there is minimal upkeep on a solar power system once the initial investment is made. The panels also shade the building, decreasing the heat load. There is no noise generated with the use of solar panels, which is a problem with generators and their corresponding noise limitations at a property line.

Solar energy prediction is typically obtained with estimated weighted moving average (EWMA) models, because of its relative consistency and periodic patterns. As long as the weather conditions remain consistent within a period, the prediction is accurate, but becomes inaccurate, with mean error over 20 percent, with frequent weather changes. Recent work utilizing small-scale solar generation uses a weather-conditioned moving average (WCMA), taking into account the mean value across days and a measured factor of solar conditions in the present day relative to previous days. While this work provides only a single future interval of prediction, it specifically addresses inconsistent conditions, with a mean error of under 10 percent.

3.1.2 Wind Power

Although it is not necessarily a reliable source of primary power, wind power can help reduce energy costs by supplementing a main power source. Moreover, it has the lowest cost of renewable technologies. A 53-meter rotor diameter is capable of producing 1 MW of power, and it takes up considerably less property than a photovoltaic system of the same capacity. However, wind turbines may not be permitted in some locations. In addition, the system still requires supplemental energy storage or the ability of feeding to and from the grid, as well as a back-up generator.Wind energy prediction can be separated into two major areas: time-series analysis of power data and wind speed prediction and conversion into power.

4 Electricity Pricing

Electricity prices change by time and by location. The frequency of the changes depend on the pricing policy of the power grid provider. The main factors[9] which determine the electricity price are the power demand,the power generation capabilities and the cost and the types [30] of the power generation. Table 2 shows the electricity prices in some countries and the date of the last modification. Electricity markets of some countries like US and Canada change the electricity price frequently (in the order of hour) while Greece changes the electricity price almost once in a year and Argentina has the same electricity price since 2006. This is depicted by the second column of the table, in which the countries with frequent flactuations in price has a price spectrum (min price,max price).

Figures 14 and 15 show the power demand (MW) and the energy price (canadian dollars/MWh)[2] on the Ontario, Canada [11].

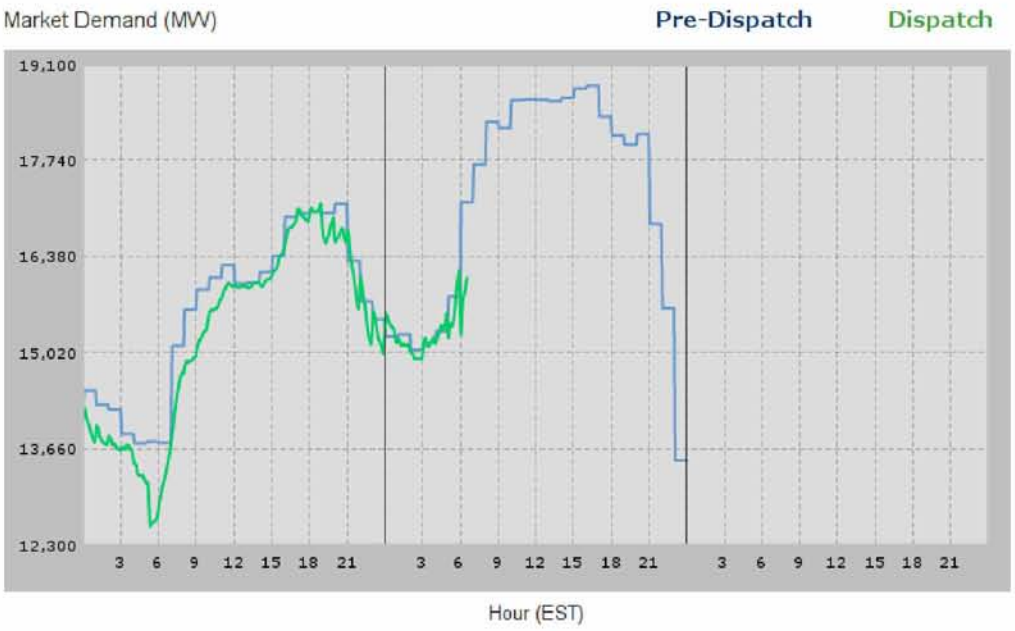

Figure 14: Three-day (9-11 July) view of market demand in Ontario Canada.

[2]One canadia dollar equals to 0.73 euros or 0.96 US dollars

Figure 15: Three-day (9-11 July) view of Energy price in Ontario Canada.

Country/Territory	US cents/kWh	As of
Argentina	5.75	2006
Australia	22-46.56	23/8/2012
Belgium	29.08	1/11/2011
Brazil	34.20	1/1/2011
Canada	6.3 - 11.8	1/11/2012
Chile	23.11	1/1/2011
Denmark	40.38	1/11/2011
Dubai	7.62	2011
Finland	20.65	1/11/2011
France	19.39	1/11/2011
Germany	31.41	31/5/2012
Hungary	23.44	1/11/2011
Hong Kong	12.02-24.00	1/4/ 2013
India	8 to 12	1/2/ 2013
Ireland	28.36	1/11/ 2011
Israel	18	1/6/ 2013
Italy	28.39	1/11/ 2011
Japan	20-24	31/12/ 2009
Latvia	18.25	1/6/2012
Malaysia	7.09-14.76	1/4/2013
Mexico	19.28	22/8/ 2012
Netherlands	28.89	1/11/ 2011
New Zealand	19.15	19/4/2012
Portugal	25.25	1/11/2011
Russia	1.7-9.58	1/ 1/ 2012
Serbia	3.93-13.48	28/2/2013
Singapore	21.53	1/4/2013
Spain	22.73	1/7/2012
South Africa	8-16	5/11/2012
Sweden	27.10	1/11/2011
Taiwan	7-17	1/6/2012
Turkey	13.1	1/6/2011
United Kingdom	20.0	30/11/2012
United States	8-17	1/9/2012

Table 2: Current electricity prices of some countries[8]

5 Service Level Agreements and Pricing Mechanisms in cloud computing

5.1 Pricing Mechanisms in cloud computing

One of the basic characteristics of cloud computing is the pay-per-use pricing models (or pay-as-you-go as introduced in section 1.2). However, not all the cloud providers follow the same pricing model. For example, some providers will find that their brand and reputation will allow them to price based on value delivered from their services, rather than purely based on hourly usage rates.

In general, cloud providers are adopting a variety of pricing mechanisms, including usage-based fixed pricing, usage-based dynamic pricing, subscription-based pricing, reserved services contracts with a combination of usage-based fixed pricing and up-front fees, and auction-based pricing.

According to [29] a representative set of pricing factors is the following : {Service Instance Type, Unit price of usage, Total Usage, Reservation Period, Reservation Fee, Support Type, Support Charge, Total Outage, Compensation}

There are two basic strategies for any pricing mechanism. One is primarily based on how much a given service costs to cloud provider to deliver it and the other is mostly determined by an estimation of current prices for cloud services in your market. Both approaches have strengths and weaknesses. The advantages of the cost-based approach are that it significantly reduces the chances of setting prices too low to earn a profit and gives a precise control over margins. On the other hand cloud provider may either left money on the table or be not competitively viable.

Market-based pricing schemes focus less on what services cost to deliver than on what cloud provider can realistically charge for them based on typical rates for similar offerings in the area. Of course, finding out what nearby providers are charging for cloud services isnt always easy. However, some firms post their rates online. Also valuable insights can be gained by participating in peer groups and membership associations or simply by asking potential customers about competing proposals.

5.2 Service Level Agreements in cloud computing

A service level agreement (SLA) is a contract where a service is formally defined [12]. This contract is conducted by two parts, the client and the service provider. A typical SLA of a cloud provider has the following components:

- **Service guarantee** specifies the metrics which a provider strives to meet over a service guarantee time period. Failure to achieve those metrics will result in a service credit to the customer. Availability, response time, disaster recovery and fault resolution time are examples of service guarantees.
- **Service guarantee time period** describes the duration over which a service guarantee should be met.
- **Service guarantee granularity** describes the resource scale on which a provider specifies a service guarantee (i.e per service, per data center, per instance, per transaction).
- **Service guarantee exclusions** are the instances that are excluded from service guarantee metric calculations. These exclusions typically include abuse of the system by a customer, or any downtime associated with the scheduled maintenance.
- **Service credit** is the amount credited to the customer or applied towards future payments if the service guarantee is not met. The amount can be a complete or a partial credit of the customer payment for the affected service.
- **Service violation measurement and reporting** describes how and who measures and reports the violation of service guarantee, respectively.

Figures 16 and 17 show a SLA comparison for some-well known cloud services. CB is an abbreviation for customer bill.

	Amazon EC2	Azure Compute	Rackspace Cloud Servers	Terremark vCloud Express	Storm on Demand
Service guarantee	Availability	Availability	Availability	Availability	Availability
Service granularity guarantee	Data center	Aggregate across all roles	Per instance*	Data center + management stack	Per instance*
Infrastructure scheduled maintenance	Unclear if excluded	Included in service guarantee	Excluded from service guarantee	Unclear if excluded	Excluded from service guarantee
OS/software patches on compute instances	N/A	Excluded from service guarantee if managed	Excluded from service guarantee if managed	N/A	Excluded from service guarantee if managed
Service guarantee time period	365 days or since last claim	Billing month	Billing month	Calendar month	Calendar month
Service credit	10% of CB if < 99.95%	10% of CB if < 99.95% 25% of CB if < 99%	5% of CB for every 30 minutes of downtime up to 100%	$1 for 15 minute downtime up to 50% of CB	1000% for every hour of downtime up to CB
Service violation reporting onus	Customer	Customer	Customer	Customer	Customer
Service violation incident reporting	N/A	5 days of incident occurence	N/A	N/A	N/A
Service violation claim filing	within 30 business days of the last reported incident in claim	within one billing month	within 30 days of downtime	within 30 days of the last reported incident in claim	within 5 days of incident in question
SLA publish date	October 23, 2008	April 9, 2010	June 23, 2009	August 31, 2009	Unknown
Credit applied towards future payments only	Yes	No	No	Yes	No

Figure 16: SLA comparison of some well-known IAAS (compute) cloud services. *implied from SLA [23]

	Amazon S3	**Azure Storage**	**Rackspace Cloud Files**
Service guarantee	Completed transactions (with no error response)	Completed transactions (within stipulated time)	Completed transactions, availability
Service granularity guarantee	Per transaction	Per transaction	Per transaction, data center
Service guarantee time period	Billing month	Billing month	Per month
Service credit	10% of CB if $<$ 99.9%, 25% of CB if $<$ 99%	10% of CB if $<$ 99.9%, 25% of CB if $<$ 99%	10% of CB if $<$ 99%, 100% of CB if $<$ 96.5%
Service violation reporting onus	Customer	Customer	Customer
Service violation incident reporting	N/A	5 days of incident occurrence	N/A
Service violation claim filing	within 10 business days following the month in which the incident occurred	within one billing month	within 30 days following unavailability
SLA publish date	October 1, 2007	November 12, 2010	June 23, 2009
Credit applied towards future payments only	Yes	No	No

Figure 17: SLA comparison of some well-known STAAS (storage) cloud services [23]

6 Problem Description

In this section, we provide an abstraction of the architecture of our system and also write down the characteristics of our problem. Cloud computing systems facilitate flexible task execution by allocating resources (CPU, storage, bandwidth, etc) on demand. The cloud provider is responsible for managing the infrastructure through efficient resource allocation and task request scheduling for virtual machines. Depending on the knowledge of the cloud provider about the parameters and the stochasticity of the system, scheduling and allocation algorithms serve the service requests in such a way to optimize an objective like the operational costs or the power consumption.

Nowadays, a big portion of the energy consumed by end-users is shifted to the core network of the cloud server infrastructure. This fact motivates our work in greening the cloud infrastructure by using renewable energy resources (RESs) attached to the server facilities. In more detail, cloud server facilities require high energy depending on the prevailing environmental conditions and the dynamic load of tasks under execution. As a general observation [33], a significant portion of the power consumption of legacy data centers is associated with the cooling needs and losses of the power provisioning equipment. The rest of the percentage above is described in a relationship to the served traffic due to the existing power proportionality of CPU and router devices [22].

An environmental friendly solution to meet the energy demands of the cloud server facilities is the deployment of RESs. The RESs reduce the dependence of the provisioned power from the main power grid and hence, if they are appropriately exploited, they can lead to a significant cost reduction. The main advantages of RESs are that they incorporate an initial capital expenditure for the purchase and deployment but they have very low operational expenses (mainly maintenance) and thus in the long-run their provisioned energy cost is very low. However, RESs have unpredictable behavior and provide a time-varying output power that in some cases is insufficient to support the operation of the cloud server facility, if the load in the latter is high enough. In this case, the required power will have to be provided by the main power grid in a price that also fluctuates with time.

We study the problem of optimal VM allocation in a set of cloud server facilities. Our objective is to minimize the total cost paid by the cloud provider to the main grid in order to support the system with the necessary power to carry out VM execution. The set of VMs, under execution at a particular server, form the server load and affect the power consumption of the components of that server, and primarily that owing to the CPU utiliza-

tion. Our system consists of a cloud provider who owns some geographically distributed cloud server facilities and exposes this infrastructure to possible customers who do not have the resources to execute their tasks. Whenever a customer asks for some available resources, the cloud provider creates a VM and selects one of his cloud server facilities to host it. All the requests initially arrive at a central dispatcher (e.g., a web server) and each request is determined by a number of **fl**oating-point **op**erations (FLOPs) that have to be executed before a deadline

For every task request cloud provider conduct a SLA with the respective client (task request owner). That SLA looks like the one in table 3.

components	value
Service guarantee	Availability
Service guarantee time period	Until the deadline
Service guarantee granularity	Cloud server facility
Service guarantee exclusions	-
Service credit	100% of customer bill
Service violation measurement and reporting	Cloud provider

Table 3: SLA of our architecture

The predefined SLA states that the cloud provider guarantees that the service will be available until the imposed at the request deadline. At least one cloud server facility would be available to execute the request and in the case of not available cloud server facility the client will take her money back.

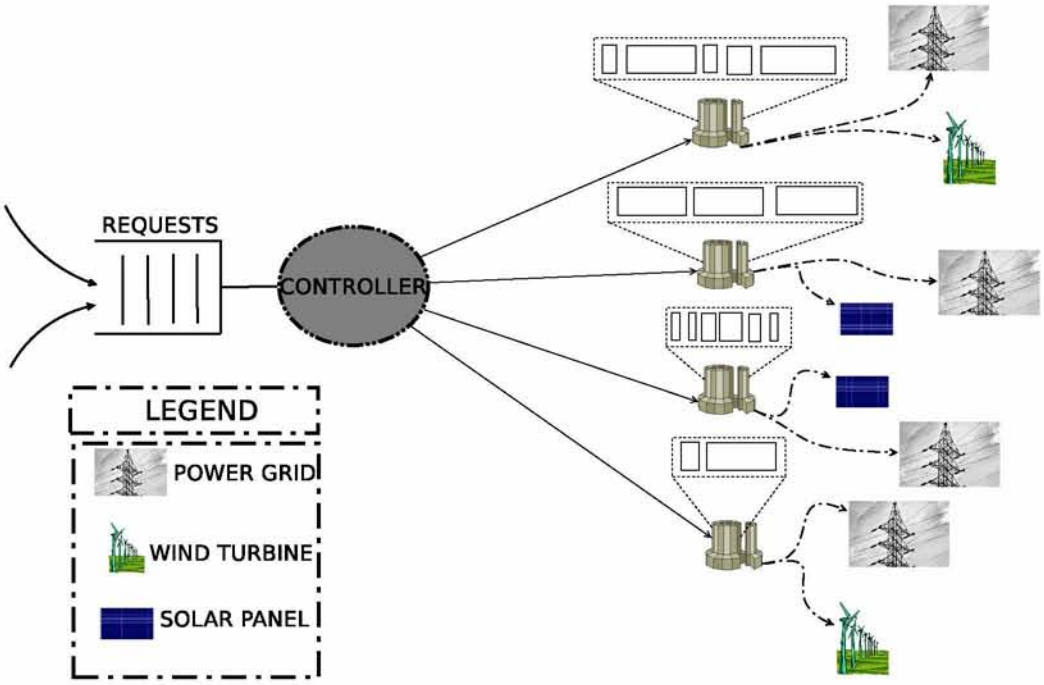

Figure 18: Requests arrive at a central controller which allocates them to cloud server facilities, each of which fulfills the energy demands from a RES and the power grid.

7 System Model

In this section, we describe in more detail the characteristics and the dynamics of the system architecture.

Symbol	Meaning	Unit
$\mathcal{D}$	The set of the cloud server facilities	Natural Number
B_i	The total processing capacity of cloud server facility i	FLOPs/second
F_j	The amount of FLOPs in the VM request j	FLOPs
d_j	The deadline imposed in the VM request j	second
c_j	The processing capacity of VM request j	FLOPs/second
α_j	The arrival time of VM request j	second
τ_j	The running time of VM request j	second
h_j	The cloud server facility which executes VM request j	Natural Number
$R_i(t)$	The power generation of the RES plugged in cloud server facility i	Watt
$p_i(t)$	The price of the power from the power grid in cloud server facility i	Euros
$L_i(t)$	The load of the cloud server facility i	FLOPs/second
PUE_i	The Power Utilization Efficiency of the cloud server facility i	real number ≥ 1
δ	The duration of one time slot	second
e_j	The execution cost of request j	Euros
m_j	The money the cloud provider will be paid for the execution of request j	Euros

Table 4: Notation Table

We consider a set $\mathcal{D}$ of D geographically distributed cloud server facilities $\mathcal{D} = \{1, \ldots, D\}$ and a set $\mathcal{B}$ of B resources, $\mathcal{B} = \{1, \ldots, B\}$. The set of the resources may include processing capacity, storage, memory, bandwidth etc. For example, a request for a database service like the Amazon relational database service [1], needs processing capacity (expressed as Elastic Compute Units), storage (GByte), memory (GByte) and I/O capacity (request rate).

In this work, we concentrate on one type of resource, the processing capacity, for the reason that the CPU is the major component that consumes

power, but our approach can be extended to include more resources. Assume that B_i is the processing capacity of cloud server facility i, in **fl**oating-point **o**perations **p**er **s**econd (FLOPS) [10].

7.1 Service Requests

Each VM request j arrives at the central queue at time α_j and is specified by a number of (FLOPs) F_j and a deadline d_j by which the execution of the VM should be finished. The cloud provider is responsible to create a VM for that request and allocate it to a cloud server. The created VM will have processing capacity c_j (FLOPS) and will be hosted for a time τ_j.

$$c_j = \frac{F_j}{\tau_j} \tag{1}$$

Note that the unit of F_j is floating-point operations (FLOPs), whereas the unit of c_j is floating-point operations per second (FLOPs/sec). The cloud provider can either allocate a VM with low processing capacity yet enough to finish the task before the deadline, or it can allocate a VM with a high processing capacity to finish the task as soon as possible. This flexibility is depicted in figure 19.

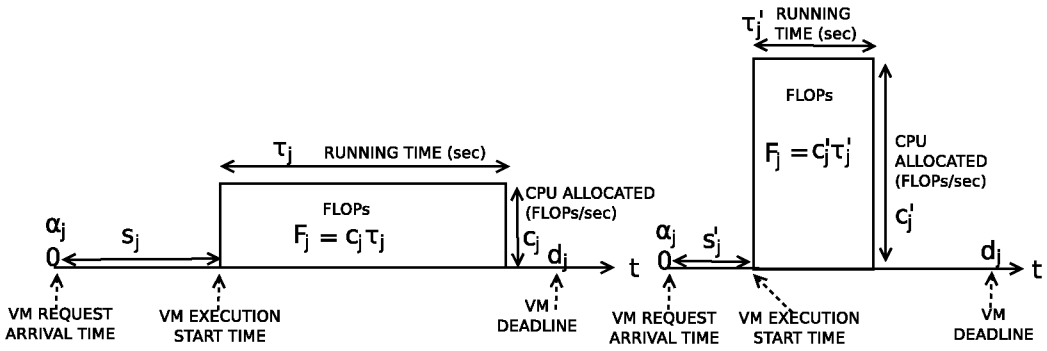

Figure 19: Two feasible realizations of processing capacity, running time and start execution time of a VM request with $\alpha_j = 0$

A SLA like that on table 3 is conducted between the customer and the cloud provider. Also cloud provider is paid m_j euros for that execution.

7.2 Model of Renewable Source Generation and Power Consumption at the Servers

Each cloud server facility $i \in \mathcal{D}$ has a RES installed that produces a time-varying power at time t, denoted by $R_i(t)$. The amount $R_i(t)$ can be known only for a short time period. This knowledge is usually extracted from weather forecast data. Each cloud server facility i can draw its energy either from the RES or from the main power grid. For the latter we assume a time-varying price per unit of power at each time t, denoted by $p_i(t)$. Also, each cloud server facility i has a load $L_i(t)$ at time t that creates power needs $f_i(L_i(t))$

where $f_i(\cdot)$ is an one-to-one function that maps the load to power consumption.

Current cloud server facilities consist of power-saving servers that incorporate frequency and voltage reduction in order to decrease the power consumption [24]. The combination of a specific CPU frequency and voltage is known as a performance state. The transition from one performance state to another does not causes performance degradation since it occurs in microseconds. These power-saving features adjust the power consumption of an idle server to its actual load and characterize the relationship of the power consumption and the load of a server, usually as a quadratic function (figure 13).

$$f_i(L_i(t)) = PUE_i \left[P_{idle}^{(i)} + \alpha \left[L_i(t) \right]^2 \right], \alpha > 0 \tag{2}$$

Figure 20 shows the power needs of some valid virtual machine allocations of one request. The power consumption of the cloud server facility remains stable over the virtual machine execution (blue squares). The power supply from the renewable energy resource is depicted by the green curve. In the case of zero load, cloud server facility consumes P_{idle} power while when it is fully loaded it consumes P_{peak}.

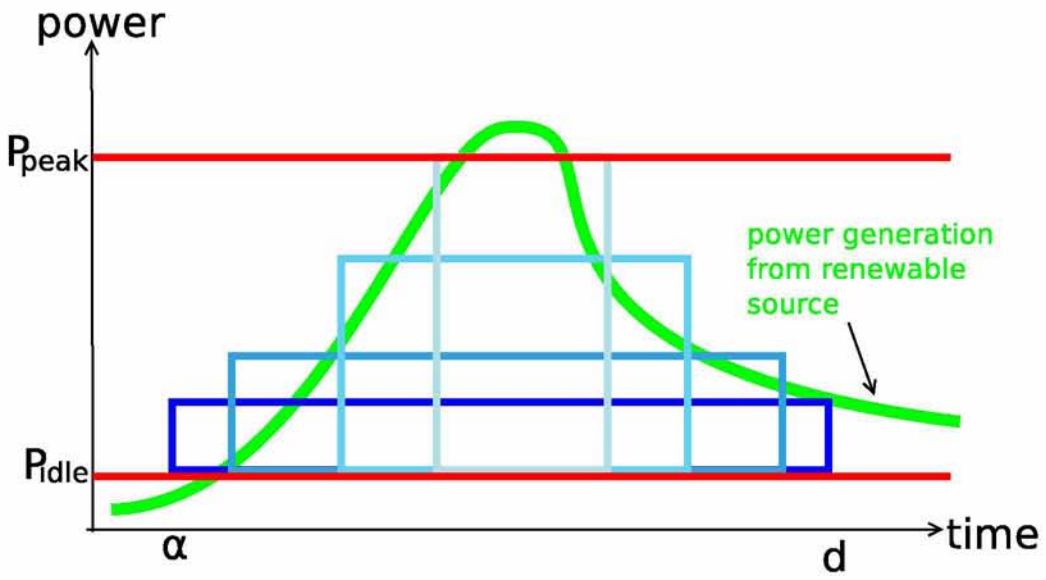

Figure 20: Valid virtual machine allocations of one request in an empty cloud server facility.

8 Problem Formulation and System Controls

In this section, we provide a mathematical formulation for the described problem of the previous two sections 6 and 7

We are given a set of requests $\mathcal{Y} = \{1, \ldots, N\}$ in a time horizon H and for each request $j \in \mathcal{Y}$, the arrival time α_j, the processing requirements (FLOPs) F_j and the deadline d_j. We want to find the start execution time s_j, the running time τ_j and the processing capacity c_j for each created VM. The set of all the feasible solutions is:

$$\mathcal{F}_j = \{(s_j, \tau_j, c_j) : \alpha_j \leq s_j \leq d_j - \tau_j \text{ and } c_j \tau_j = F_j\} \tag{3}$$

It is worth mentioning that once a VM starts its execution in a cloud server facility, it is impossible to migrate to another cloud server facility or to fluctuate its processing capacity.

8.1 One Cloud Server Facility

In the case of a single cloud server facility, the price of power grid per unit of power is described by $p(t)$ and the RES power generation by $R(t)$. Also the load is described by:

$$L(t) = \sum_{j: s_j \leq t \leq s_j + \tau_j} c_j \tag{4}$$

and the total power consumption by $f(L)$.

The optimal solution for every request in $\mathcal{Y}$ is given by:

$$\min_{(s_j, \tau_j, c_j) \in \mathcal{F}_j} \int_0^H p(t) \left[f(L(t)) - R(t) \right]^+ dt \tag{5}$$

where q^+ equals q if q is greater than zero otherwise equals zero.

8.2 Multiple Cloud Server Facilities

In the case of more than one cloud server facility we have to introduce a binary parameter x_{ij} indicating the selection of the server facility i to execute the VM j, $\underline{x} = (x_{ij} : i = 1, \ldots, D, j = 1, \ldots, N)$. Since, each VM can be assigned only in one cloud server facility, the assignment variable is described by the following equation:

$$x_{ij} = \begin{cases} 1 & \text{if VM } j \text{ assigned to cloud server facility } i \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

The load of every cloud facility i is described by:

$$L_i(t) = \sum_{j:s_j \leq t \leq s_j+\tau_j \text{ and } x_{ij}=1} x_{ij} c_j \tag{7}$$

The optimal solution for every request in $\mathcal{Y}$ is given by:

$$\min_{\substack{(s_j,\tau_j,c_j)\in\mathcal{F}_j \\ \underline{x}}} \sum_{i=1}^{D} \int_0^H p_i(t) \left[f_i(L_i(t)) - R_i(t) \right]^+ dt \tag{8}$$

subject to:

$$\sum_{i=i}^{D} x_{ij} = 1 \ \ \forall j \tag{9}$$

8.3 Alternative Formulation

Since the control variables of our problem are related to every request $j \in \mathcal{Y}$ it is worth providing an alternative formulation of the same problem which shows directly the control variables. In more detail equation 10 is equivalent to 5 and equation 11 is to 8 and 9.

$$\min_{(s_j,\tau_j,c_j)\in\mathcal{F}_j} \sum_{j\in\mathcal{Y}} \int_{s_j}^{s_j+\tau_j} p(t) \left[[f(L(t)+c) - R(t)]^+ - [f(L(t)) - R(t)]^+ \right] dt \tag{10}$$

$$\min_{\substack{(s_j,\tau_j,c_j)\in\mathcal{F}_j \\ h_j\in\{1,\ldots,D\}}} \sum_{j\in\mathcal{Y}} \int_{s_j}^{s_j+\tau_j} p_{h_j}(t) \left[\left[f_{h_j}(L_{h_j}(t)+c) - R_{h_j}(t) \right]^+ - \left[f_{h_j}(L_{h_j}(t)) - R_{h_j}(t) \right]^+ \right] dt \tag{11}$$

where h_j is the cloud server facility that executes the VM of the request j.

The lagrangian function of 10 is:

$$J = \sum_{j\in\mathcal{Y}} \int_{s_j}^{s_j+\tau_j} p(t) \left[[f(L(t)+c) - R(t)]^+ - [f(L(t)) - R(t)]^+ \right] dt \tag{12}$$

$$- \lambda_1(\alpha_j - s_j) - \lambda_2(s_j + \tau_j - d_j) - \mu(\tau_j - \frac{F_j}{c_j}) \tag{13}$$

In order to find the optimal values for s_j, τ_j and c_j we have to calculate the partial derivative for every control variable.

Math Reminder 1. *Let $f(x,t)$ be a function such that both $f(x,t)$ and its partial derivative $f_x(x,t)$ are continuous in t and x in some region of the (x,t)-plane, including $a(x) \leq t \leq b(x), x_0 \leq x \leq x_1$. Also suppose that the functions $a(x)$ and $b(x)$ are both continuous and both have continuous derivatives for $x_0 \leq x \leq x_1$. Then for $x_0 \leq x \leq x_1$ [7]:*

$$\frac{d}{dx}\left(\int_{a(x)}^{b(x)} f(x,t)dt\right) = f(x,b(x))b^{'}(x) - f(x,a(x))a^{'}(x) + \int_{a(x)}^{b(x)} f_x(x,t)dt$$

and

$$\frac{\partial}{\partial b}\left(\int_a^b f(x)dx\right) = f(b) \tag{14}$$

Using 14 for s_j:

$$\begin{aligned}
\frac{\partial J}{\partial s_j} &= \frac{\partial}{\partial s_j}\int_{s_j}^{s_j+\tau_j} p(t)\left[[f(L(t)+c)-R(t)]^+ - [f(L(t))-R(t)]^+\right]dt + \lambda_1 - \lambda_2 \\
&= \frac{\partial}{\partial s_j}\int_{0}^{s_j+\tau_j} p(t)\left[[f(L(t)+c)-R(t)]^+ - [f(L(t))-R(t)]^+\right]dt - \\
&\frac{\partial}{\partial s_j}\int_{0}^{s_j} p(t)\left[[f(L(t)+c)-R(t)]^+ - [f(L(t))-R(t)]^+\right]dt + \lambda_1 - \lambda_2 \\
&= p(s_j+\tau_j)\left[[f(L(s_j+\tau_j)+c)-R(s_j+\tau_j)]^+ - [f(L(s_j+\tau_j))-R(s_j+\tau_j)]^+\right] - \\
&p(s_j)\left[[f(L(s_j)+c)-R(s_j)]^+ - [f(L(s_j))-R(s_j)]^+\right] + \lambda_1 - \lambda_2
\end{aligned}$$

In order to find the optimal value we set equal to zero the partial derivative $\frac{\partial J}{\partial s_j} = 0$ and we take:

$$\begin{aligned}
&p(s_j+\tau_j)\left[[f(L(s_j+\tau_j)+c)-R(s_j+\tau_j)]^+ - [f(L(s_j+\tau_j))-R(s_j+\tau_j)]^+\right] \\
&- p(s_j)\left[[f(L(s_j)+c)-R(s_j)]^+ - [f(L(s_j))-R(s_j)]^+\right] \\
&= \lambda_2 - \lambda_1
\end{aligned}$$

Similarly, for τ_j,

$$\begin{aligned}
\frac{\partial J}{\partial \tau_j} &= \frac{\partial}{\partial \tau_j}\int_{s_j}^{s_j+\tau_j} p(t)\left[[f(L(t)+c)-R(t)]^+ - [f(L(t))-R(t)]^+\right]dt - \lambda_2 - \mu \\
&= p(s_j+\tau_j)\left[[f(L(s_j+\tau_j)+c)-R(s_j+\tau_j)]^+ - [f(L(s_j+\tau_j))-R(s_j+\tau_j)]^+\right] \\
&- \lambda_2 - \mu
\end{aligned}$$

In order to find the optimal value we set equal to zero the partial derivative $\frac{\partial J}{\partial \tau_j} = 0$ and we take:

$$
\begin{aligned}
&p(s_j + \tau_j) \left[[f(L(s_j + \tau_j) + c) - R(s_j + \tau_j)]^+ - [f(L(s_j + \tau_j)) - R(s_j + \tau_j)]^+ \right] \\
&- p(s_j) \left[[f(L(s_j) + c) - R(s_j)]^+ - [f(L(s_j)) - R(s_j)]^+ \right] \\
&= \lambda_2 + \mu
\end{aligned}
$$

For c_j we use equation 1

9 Offline Algorithm

In this section we try to find an offline algorithm to solve the virtual machine allocation problem. More specifically, given the set:

$$\mathcal{Y} = \{(\alpha_1, F_1, d_1), (\alpha_2, F_2, d_2), \ldots, (\alpha_N, F_N, d_N)\} \tag{15}$$

of all the request in a time horizon H, we try to find the start execution time s_j, the execution length τ_j, the processing capacity c_j and the cloud server facility h_j that will host the virtual machine of request j of every request $j \in \mathcal{Y}$.

9.1 Slotted Time

Since the predefined optimization problems in section 8 can not be solved in continuous time we make the assumption the the system is slotted and that every event occurs in a timestamp that is multiple of δ. In that case, we should modify the predefined equations according to the following rule:

$$\int_{t_a}^{t_b} (expression\_of\_t)\, dt \longrightarrow \sum_{t=t_a}^{t_b} (expression\_of\_t) \tag{16}$$

9.2 One Cloud Server Facility

In the case of one cloud server facility we have to solve the following problem:

$$\min_{\boldsymbol{s},\boldsymbol{\tau},\boldsymbol{c}} \int_0^H p(t) \left[f(L(t)) - R(t)\right]^+ dt \tag{17}$$

subject to:

$$L(t) = \sum_{j:s_j \le t \le s_j+\tau_j} c_j \tag{18}$$

9.2.1 Only One Request Example

$$\min_{(s_j,\tau_j,c_j)\in\mathcal{F}_|} \int_0^H p(t)\left[f(c_j)-R(t)\right]^+ dt \tag{19}$$

$$\min_{(s_j,\tau_j,c_j)\in\mathcal{F}_|} \int_{\alpha_j}^{d_j} p(t)\left[f(c_j)-R(t)\right]^+ dt \tag{20}$$

$$\min_{(s_j,\tau_j,c_j)\in\mathcal{F}_|} \int_{s_j}^{s_j+\tau_j} p(t)\left[f(c_j)-R(t)\right]^+ dt \tag{21}$$

$$\min_{(s_j,\tau_j,c_j)\in\mathcal{F}_|} \sum_{t=s_j}^{s_j+\tau_j} p(t)\left[f(c_j)-R(t)\right]^+ \tag{22}$$

The execution time of the request could be any multiple of δ in the following spectrum:

$$\tau_j \in \left\{\frac{F_j}{B},\ldots,\frac{d_j-\alpha_j}{\delta}\right\} \tag{23}$$

under the constraint:

$$L(t)\leq B \quad \forall t\in[s_j,s_j+\tau_j] \tag{24}$$

Given the execution time we can calculate the processing capacity $c_j=\frac{F_j}{\tau_j}$ and thereore we can determine the hosting cost for every possible s_j,τ_j and c_j. In general:

$$s_j = k\cdot\delta \tag{25}$$

$$\tau_j = k^{'}\cdot\delta \tag{26}$$

$$c_j \in \{\frac{F}{\tau},\ldots,\min(B,\frac{F}{\delta})\} \tag{27}$$

where k and $k^{'}$ are any possitive integers.

The following picture shows a general case of one request with two possible processing capacities but alot of starting execution times.

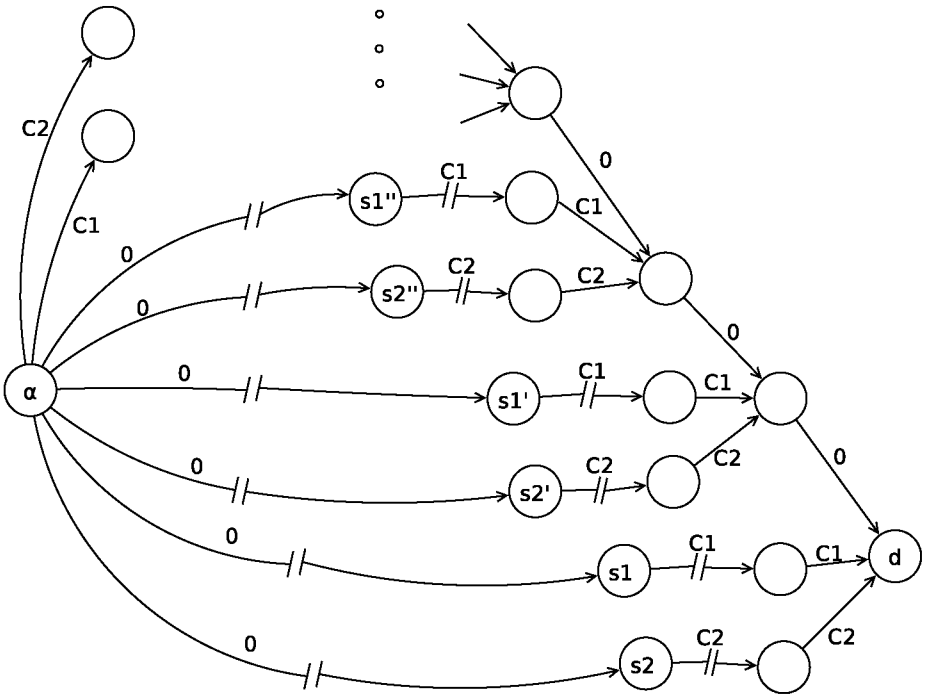

Figure 21: All possible executions of a vm with only two possible processing capacities.

9.2.2 Four Requests example

Before we present an offline algorithm which determines all the control variables for every request, we show a small example of four requests and only one cloud server facility. This example depicts the basic concepts needed to understand the offline algorithm. In general, every request j can be served by creating a virtual machine with a set of possible realizations as depicteded in figure 19.

Under the assumption that the system is slotted, we can determine the set of all the possible executions of request j, $\mathcal{W}_j = \{w_j^{(1)}, w_j^{(2)}, \ldots\}$ with size W_j. The number of all the possible executions of all the requests is:

$$W = \prod_{j=1}^{N} W_j \tag{28}$$

And the set of all the possible executions is the cartesian product

$$\mathcal{W} = \mathcal{W}_1 \times \mathcal{W}_2 \times \ldots \times \mathcal{W}_N \tag{29}$$

and can be illustrated by a tree like the one on figure 22. The red nodes is $\mathcal{W}_1$, the green is $\mathcal{W}_2$, the blue is $\mathcal{W}_3$ and the yellow is $\mathcal{W}_4$.

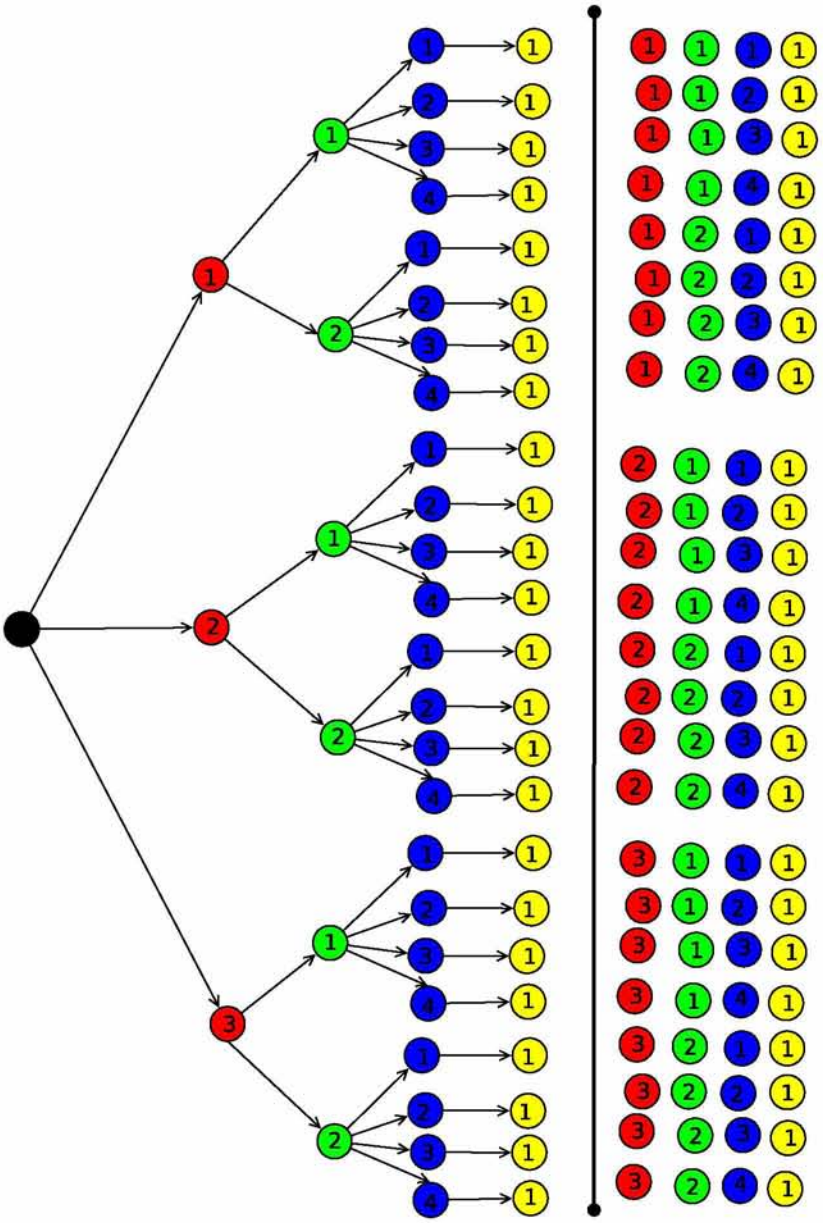

Figure 22: all possible executions of four requests with $W_1 = 3, W_2 = 2, W_3 = 4, W_4 = 1$

Every possible execution $w = (w_1, w_2, \ldots, w_N) \in \mathcal{W}$, where $w_j = (s_j, \tau_j, c_j)$ is the execution of job j ,has a cost equal to:

$$\sum_{t=s_1}^{d_N} p(t) \left[f(L(t)) - R(t) \right]^+ + \infty \left[L(t) - B \right]^+ \tag{30}$$

subject to:

$$L(t) = \sum_{j: s_j \leq t \leq s_j + \tau_j} c_j \tag{31}$$

We assume that:

$$\infty \left[L(t) - B \right]^+ = \begin{cases} \infty & \text{if } L(t) > B \\ 0 & \text{otherwise.} \end{cases} \tag{32}$$

Practically, if in one execution the load of the cloud server facility exceeds the processing capacity of the cloud server facility the cost of this execution is infinity because it is impossible to happen.

9.2.3 Multiple cloud server facilities

In that case $\mathcal{W}_j$ has multiple times, as many as the number of the cloud server facilities, the same executions but for different cloud server facility. In more detail, w_j has one more entry which is the number of the cloud server facility. i.e $w_j = (s_j, \tau_j, c_j, h_j)$. Also the execution cost is:

$$\sum_{i=1}^{D} \sum_{t=\alpha_1}^{d_N} p_i(t) \left[f_i(L_i(t)) - R_i(t) \right]^+ + \infty \left[L_i(t) - B_i \right]^+ \tag{33}$$

subject to:

$$L_i(t) = \sum_{j: s_j \leq t \leq s_j + \tau_j} c_j 1_{i=h_j} \tag{34}$$

where

$$1_{i=h_j} = \begin{cases} 1 & \text{if } i = h_j \\ 0 & \text{otherwise.} \end{cases} \tag{35}$$

$$\infty \left[L_i(t) - B_i \right]^+ = \begin{cases} \infty & \text{if } L_i(t) > B_i \\ 0 & \text{otherwise.} \end{cases} \tag{36}$$

9.3 Transform the Offline problem to a Shortest Path problem

If we add a dummy node in figure 22 which is connected to all the leaf nodes, we are taking a directed graph whose every path from the root node to the dummy node represents a possible execution. Having that in mind we add the appropriate weights to every edge of the graph in such a way to transform the offline problem proposed in section 9 into a shortest path problem[13]. After that transformation we can use one of the well-known algorithms (BellmanFord, Dijkstra, Gabow, etc) to solve the problem.

9.4 Edge Weights

Every edge from a node $w_j^{(k)} \in \mathcal{W}_j$ to a node $w_{j+1}^{(l)} \in \mathcal{W}_{j+1}$ must have weight equal to the execution cost up to the node of $w_{j+1}^{(l)}$ minus the execution cost up to the node of $w_j^{(k)}$. By doing that, we ensure that the total cost of any path from the root to the dummy node is equal to the cost needed for the execution of all the requests. The selected $w_j^{(l)}$ execution for the request j is determined by the node on the path.

$$z_{R \rightsquigarrow k}^{(1)} = \sum_{t=s_1^k}^{s_1^k+\tau_1^k} p_{h_1^k}(t) \left[f_{h_1^k}(c_1^k) - R_{h_1^k}(t) \right]^+, \quad (s_1^k, \tau_1^k, c_1^k, h_1^k) = w_1^k \tag{37}$$

$$z_{l \rightsquigarrow k}^{(j)} = \left(\sum_{i=1}^{D} \sum_{t=s_1^k}^{d_j} p_i(t) \left[f_i(L_i(t)) - R_i(t) \right]^+ \right) - z_l^{(j-1)} \quad 1 < j \leq N \tag{38}$$

$$z_{l \rightsquigarrow S}^{(N+1)} = 0 \tag{39}$$

The number of all the edges is:

$$E = \overbrace{\sum_{i=1}^{N} \prod_{j=1}^{i} W_j}^{\text{tree edges}} + \overbrace{\prod_{j=1}^{N} W_j}^{\text{edges to S}} \tag{40}$$

and the number of all the nodes is:

$$V = \sum_{i=1}^{N} \prod_{j=1}^{i} W_j \tag{41}$$

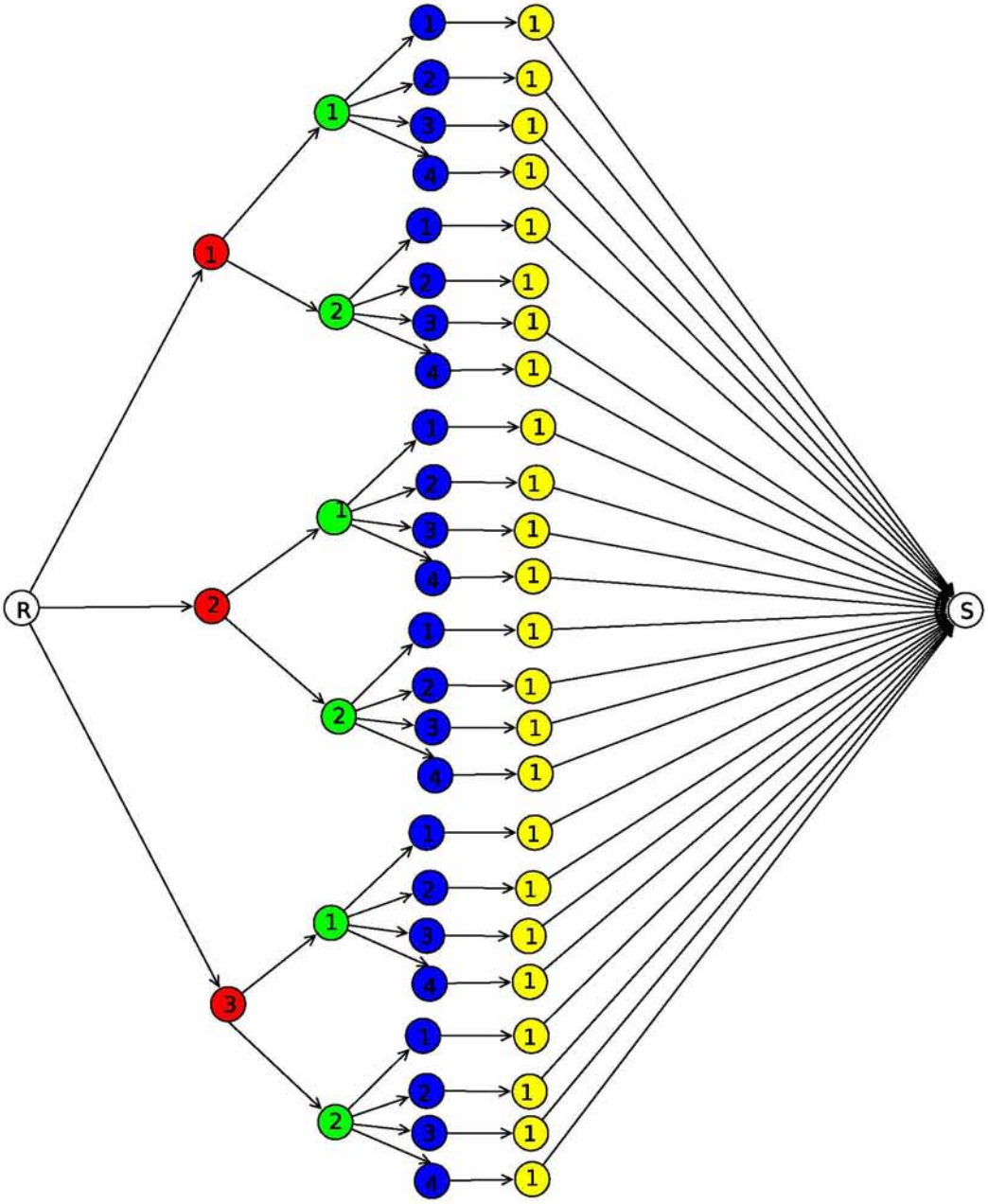

Figure 23: The graph created by the tree of all possible executions of the example in subsection 9.2.2

The number of the edges and the number of the nodes given by equations 40 and 41 can be reduced after a sorting of the set $\mathcal{Y}$ with respect to the number of possible executions of every job j, W_j. Figure 24 depicts that reduction. The average complexity of a shorting algorithm [14] is $O(NlogN)$,

where N is the number of the requests.

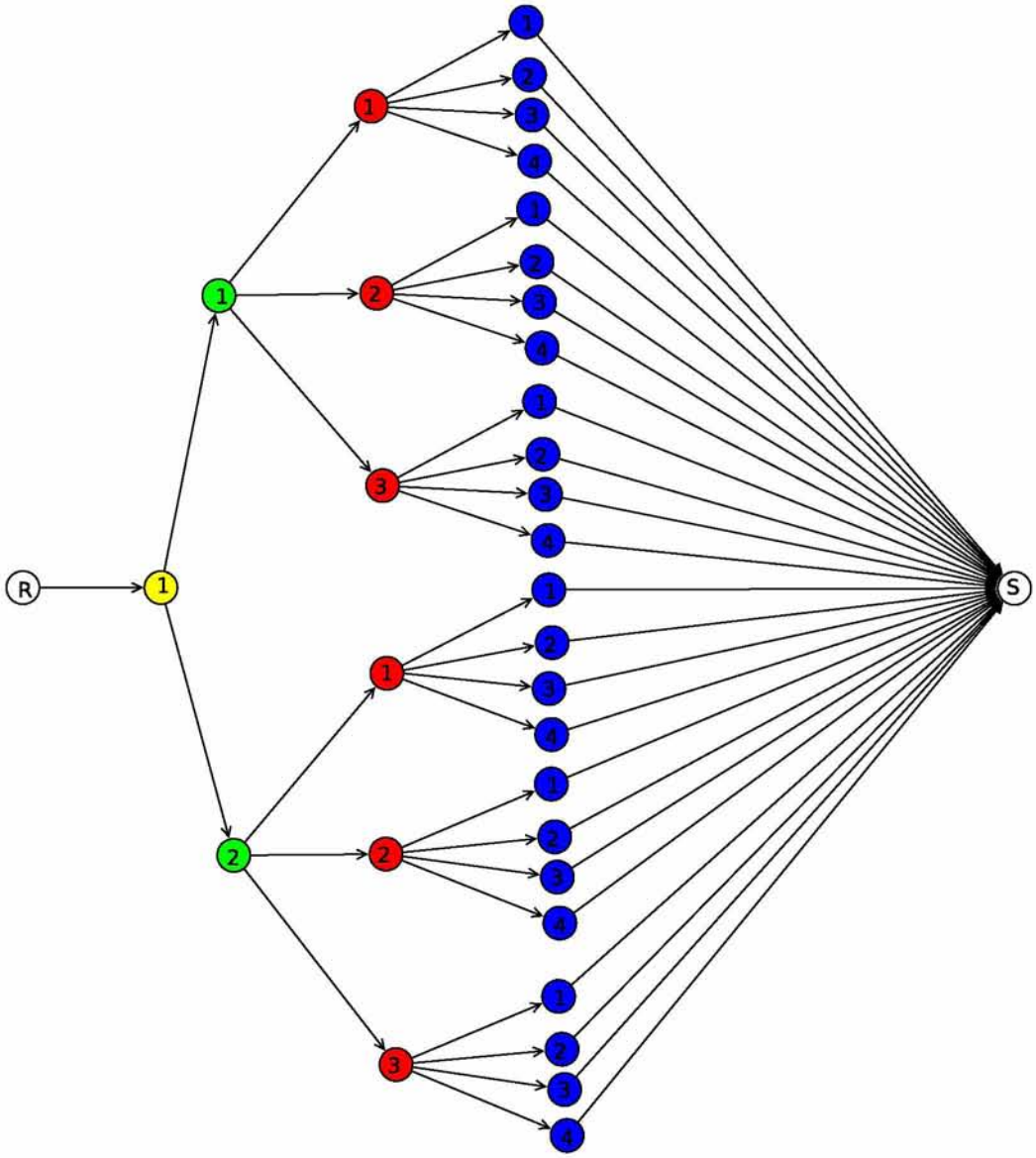

Figure 24: The graph created by the tree of all possible executions of the example in subsection 9.2.2 after the sorting by the number of all possible executions

9.5 Complexity

In order to determine the minimum cost of execution of all the requests in $\mathcal{Y}$ we have to do the following steps:

1. $\forall j \in \mathcal{Y}$ find $\mathcal{W}_j$
2. sort $\mathcal{Y}$ with respect to W_j
3. calculate $\mathcal{W}$ and create a graph similar to figure 24.
4. use equations 37-39 to assign the appropriate weight to every edge
5. use a shortest path algorithm to find the optimal w_j^* $\forall j \in Y$

The last two steps are more complex and these are the processes who determine the complexity of our solution since the first three require polynomial time. Step 4 requires a traverse of the tree (graph without the zero-cost edges to node S) created by step 3, this can be done by a Depth-first search algorithm [6] (or Breadth-first algorithm [2]) with complexity $O(E)$ while, the sortest path algorithm requires $O(V^2)$ if we use a list and $O(E + VlogV)$ if we use a fibonacci heap.

9.6 Pseudo-Code for general case

In this subsection we present the pseudo-code of the main functions of the offline algorithm. Function 1 calculates the number of all the possible executions of a given request. Function 2 uses function 1 to create a vectror with all the possible executions and function 3 calculates the size of $\mathcal{W}$. Function 5 produces a two-dimentional vector with all the possible executions. Function 6 is a recursive function which practically parses $\mathcal{W}$ and finds the cost of every senario $w \in \mathcal{W}$. Pseudo-code 8 describes the offline algorithm and functions 9 and 10 determine the minimum cost senario and the execution of every request. Figure 25 shows how the deadline and the number of flops affect the number of all the possible executions.

Algorithm 1 All possible executions of one VM request (find W_j)

```
1: function NumberOfRequestsPossibleExecutions(request j, D)
2:     number_of_possible_executions ← 0
3:     for all d ∈ D do
4:         s ← α_j
5:         while s < d_j do
6:             e ← s + δ
7:             while e ≤ d_j do
8:                 number_of_possible_executions + +
9:                 e ← e + δ
10:            end while
11:            s ← s + δ
12:        end while
13:    end for
14:    return number_of_possible_executions
15: end function
```

Algorithm 2 All possible executions

```
1: function AllPossibleExecutions(Y, D)
2:     for all r ∈ Y do
3:         Possible_executions[r] =NumberOfRequestsPossibleExecutions(r,D)
4:     end for
5:     return Possible_executions
6: end function
```

Algorithm 3 Number of all the possible executions

```
1: function NumberOfAllPossibleExecutions(𝒴)
2:     total_possible_executions = 1
3:     for all r ∈ 𝒴 do
4:         total_possible_executions∗ = Possible_executions[r]
5:     end for
6:     return total_possible_executions
7: end function
```

Algorithm 4 This function returns the start of request's possible executions in vector *total_possible_executions*

```
1: function BeforeRequestPossibleExecutions(request j,𝒴)
2:     before_req_possible_executions = 0
3:     for all r ∈ 𝒴 and r < j do
4:         before_req_possible_executions+ = Possible_executions[r]
5:     end for
6:     return before_req_possible_executions
7: end function
```

Algorithm 5 Save All Possible Executions (find $(\mathcal{W}_1, \ldots, \mathcal{W}_N)$)

```
function SaveAllPossibleExecutions(𝒴, 𝒟)
    pos ← 0
    for all r ∈ 𝒴 do
        for all i ∈ 𝒟 do
            s ← α_j
            while s < d_j do
                e ← s + δ
                while e ≤ d_j do
                    total_possible_executions[pos][0] = i
                    total_possible_executions[pos][1] = s
                    total_possible_executions[pos][2] = e
                    total_possible_executions[pos][3] = F_r/(e−s)
                    e ← e + δ
                    pos ← pos + 1
                end while
                s ← s + δ
            end while
        end for
    end for
end function
```

Algorithm 6 Find Minimum Cost senario

function MinCostSenario($request, senario, confs, temp\_load, costs$)
  if senario = NumberOfAllPossibleExecutions($\mathcal{Y}$) **then**
    FindMinimum(costs)
  else if $request = N - 1$ **then**
    $x \leftarrow$ BeforeRequestPossibleExecutions(request,$\mathcal{Y}$)
    $x \leftarrow x + confs[request]$
    $t \leftarrow 0$
    while $t \leq H$ **do**
      $s \leftarrow total\_possible\_executions[x][1]$
      $e \leftarrow total\_possible\_executions[x][2]$
      $i \leftarrow total\_possible\_executions[x][0]$
      $c \leftarrow total\_possible\_executions[x][3]$
      if $t \geq$ s and $t < e$ **then**
        $temp\_load[i][t] += c$
      end if
      $t \leq t + \delta$
    end while
    $d \leftarrow 0$
    while $d < D$ **do**
      $t \leftarrow 0$
      while $t \leq H$ **do**
        $costs[senario] += p_d(t)\,[f_d(temp\_load[i][t] - R_d(t))]^+$
        $t \leq t + \delta$
      end while
      $d \leq d + 1$
    end while
    $confs[request] \leftarrow confs[request] + 1$
    $l \leftarrow$ NumberOfRequestsPossibleExecutions($request, \mathcal{D}$)
    $confs[request] \leftarrow confs[request] \mod l$
    MinCostSenario($0, senario + 1, confs, temp\_load, costs$)

Algorithm 7 Find Minimum Cost senario (Continue)

```
    else
        x ← BeforeRequestPossibleExecutions(request,𝒴)
        x ← x + confs[request]
        t ← 0
        while t ≤ H do
            s ← total_possible_executions[x][1]
            e ← total_possible_executions[x][2]
            i ← total_possible_executions[x][0]
            c ← total_possible_executions[x][3]
            if t ≥ s and t < e then
                temp_load[i][t]+ = c
            end if
            t ≤ t + δ
        end while
        l ← NumberOfRequestsPossibleExecutions(request + 1, 𝒟)
        if confs[request + 1] = l then
            confs[request] ← confs[request] + 1
            l ← NumberOfRequestsPossibleExecutions(request, 𝒟)
            confs[request] ← confs[request]  mod l
        end if
        MinCostSenario(request + 1, senario, confs, temp_load, costs)
    end if
end function
```

Algorithm 8 Offline Algorithm

```
function OFFLINEALG(𝒴, 𝒟)
    while r ∈ 𝒴 do
        confs[r] ← 0
    end while
    while d ∈ 𝒟 do
        t ← 0
        while t ≤ H do
            temp_load[d][t] ← 0
        end while
    end while
    I ← NumberOfAllPossibleExecutions(𝒴)
    i ← 0
    while i < I do
        costs[i] ← 0
    end while
    MinCostSenario(0, 0, confs, temp_load, costs)
end function
```

Algorithm 9 Find Minimum cost senario

```
function FindMinimum(costs)
    minCost ← ∞
    optsenario ← 0
    I ← NumberOfAllPossibleExecutions(𝒴)
    i ← 0
    while i < I do
        if costs[i] < minCost then
            minCost ← costs[i]
            optsenario ← i
        end if
    end while
end function
```

Algorithm 10 Find Executions of Minimum Cost Senario

function FINDEXECS($optsenario, \mathcal{Y}$)
  $I \leftarrow$ NUMBEROFALLPOSSIBLEEXECUTIONS($\mathcal{Y}$)
  while $req \in \mathcal{Y}$ **do**
    $l \leftarrow$ NUMBEROFALLPOSSIBLEEXECUTIONSAFTER($req, \mathcal{Y}$)
    if $l \neq 1$ **then**
      $w^*_{req} = optsenario/l$
    else
      $l \rightarrow$NUMBEROFREQUESTSPOSSIBLEEXECUTIONS($req, \mathcal{Y}$)
      $w^*_{req} = optsenario \mod l$
    end if
  end while
end function

Algorithm 11 Number of all the possible executions of requests after a given request

1: **function** NUMBEROFALLPOSSIBLEEXECUTIONSAFTER($req, \mathcal{Y}$)
2: $next\_possible\_executions = 1$
3: **for all** $r \in \mathcal{Y}$ and $r > req$ **do**
4: $next\_possible\_executions* = Possible\_executions[r]$
5: **end for**
6: **return** $next\_possible\_executions$
7: **end function**

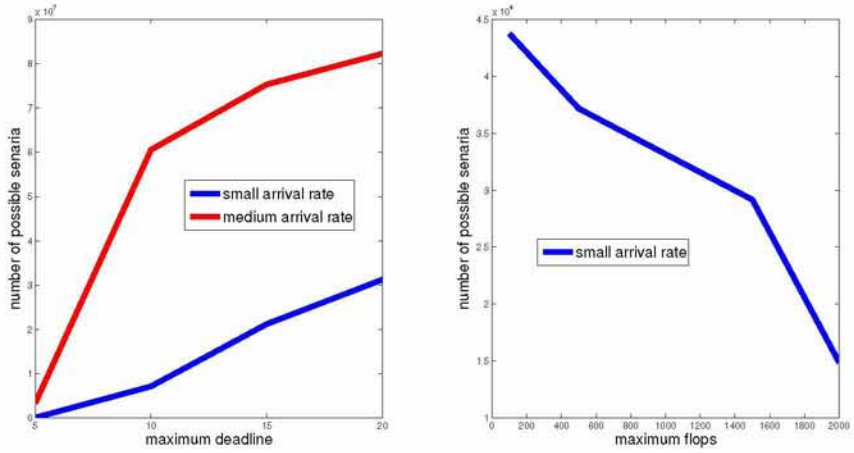

Figure 25: Number of possible executions vs the maximum flops and maximum deadline.

10 Virtual Machine Allocation Algorithms

In this section, we propose an online algorithm that operates upon arrival of a VM request and determines to which cloud server facility it will be allocated given the knowledge of the RESs power production and price of grid power.

10.1 Online Algorithm with Lookahead Window T

In this section, we describe an algorithm that defines the time a new request starts being served and the cloud server facility that executes the created VM. Furthermore, our algorithm specifies the processing capacity of the VM and the running time, given that the RES power supply and the power grid prices are known for the next small lookahead horizon $T \leq H$. This assumption is valid because the RES power supply depends on the weather conditions which can be assumed as known for a lookahead time horizon of some hours. Also, the price of the power from the power grid does not change frequently and the assumption of the apriori knowledge for the next few hours is valid. The additional power needed to host VM j in a cloud server facility i, if available resources exist, at time t equals:

$$C_i(t, c_j) = (f(L_i(t) + c_j) - R_i(t))^+ - (f(L_i(t)) - R_i(t))^+ \tag{42}$$

The additional power is zero if there is enough renewable capacity $R_i(t)$. The additional cost of hosting VM j in a cloud server facility i starting at time s_j while taking into account the price at i and the running time of the VM $c_j = \frac{F_j}{\tau_j}$, is:

$$c_{j \to i}(s_j, \tau_j) \quad = \quad \int_{s_j}^{s_j + \tau_j} p_i(t) C_i(t, \frac{F_j}{\tau_j}) dt \tag{43}$$

Given that the new VM j will be assigned to cloud server facility i, the optimal time to start s_j^*, the optimal running time τ_j^* and the optimal processing capacity c_j^* are given from:

$$\min_{(s_j, \tau_j, c_j) \in \mathcal{F}_j} c_{j \to i}(s_j, \tau_j) \tag{44}$$

For $t \in [T, d_j]$ we use estimated values of $R_i(t)$ and $p_i(t)$. Now using equation (44) we produce a vector space $(\mathbf{s}_j^*, \tau_j^*, \mathbf{c}_j^*)$ for all the cloud server facilities and select the cloud server facility with the minimum cost. Therefore the minimum additional cost for hosting the VM j is: $c_{j \to i^*}(s_j^*, \tau_j^*)$.

10.2 Online Greedy Algorithm

An extreme case, is the case of no-knowledge about the stochastic parameters of our system. An online greedy algorithm creates a VM to the cloud server facility that offers the cheapest additional hosting cost by taking into account only the instantaneous parameters. The new VM starts its execution immediately and finishes on the deadline. Then the cloud server facility i^* that will host the VM determined at the request is:

$$i^* = \arg \min_{i=[1,\ldots,D]} \left[p_d(a_j) C_d(a_j, \frac{F_j}{d_j - a_j}) \right] \tag{45}$$

10.3 Online Random Algorithm

In order to assess the performance of our proposed algorithm we compare it with a random algorithm which assigns every incoming request to a random cloud server facility that has the resources to host it. This naive algorithm is used as an upper bound benchmark in the performance of our algorithm.

$$i^* = U\left(1, D\right) \tag{46}$$

where $U(a, b)$ is the uniform distribution in $[a, b]$

11 Numerical Results of Online Algorithms

In this section we present the performance of our allocation algorithms. For this reason, we implement a simulator using Java.

11.1 Experiment One

In that experiment we use 10 geographically distributed cloud server facilities with equal capacity. The duration of the experiment is two days. Each cloud server facility is equipt by a solar panel whose peak power generation 1.2 times the maximum power consumption P_{peak}. The solar panel power generation is gaussian shaped. The price of the power from the power grid is selected by table 2. More specifically, we select the following ten locations: australia, canada, hong kong, japan,south africa, Netherlands, Mexico, Denmark, Argentina, Dubai.

Figure 26 shows how the increase of the arrival rate affect the total cost of execution. Furthermore, online algorithms with look-ahead window of 2 and 7 hours are more robust in that increase and this is shown by the smaller increase rate in their curves. Also, online algorithm with look-ahead window of 7 hours exploits better the knowledge of the stochastic parameters and responds better in the arrival rate increase.

The number of flops and the deadline follow a uniform distribution whose parameters do not change accross the experiment.

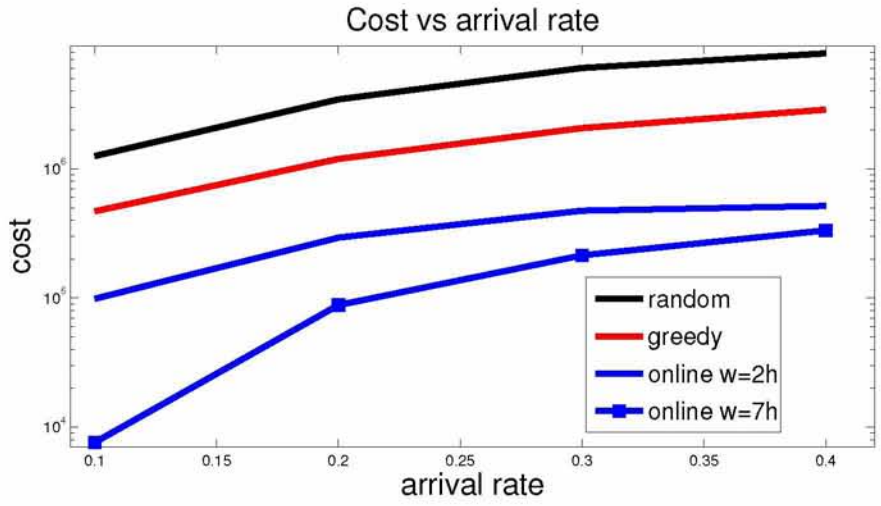

Figure 26: Total cost vs arrival rate. For every value of the arrival rate we conduct the same experiment 1000 times.

Figure 27 shows how the increase of the maximum number of flops affect the total cost of execution. Again, online algorithm with look-ahead window of 7 hours exploits better the knowledge of the stochastic parameters and responds better in the increase of the flops number.

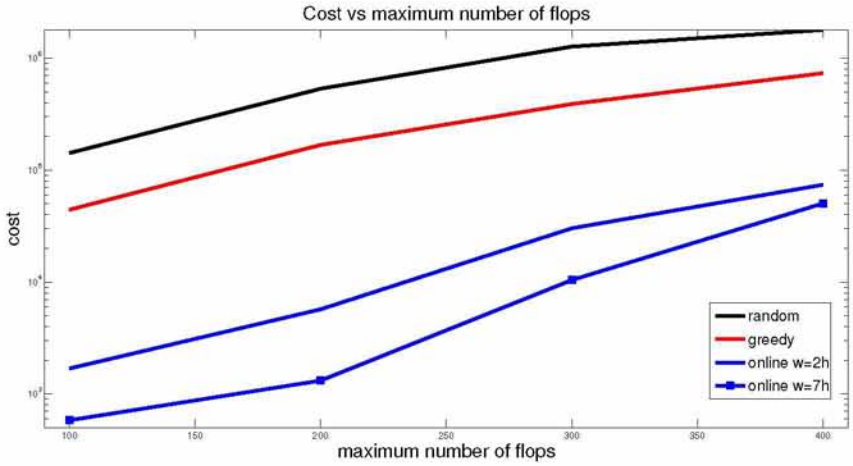

Figure 27: Total cost vs arrival rate. For every value of the maximum flops number we conduct the same experiment 1000 times.

11.2 Experiment Two

The setting of that experiment consists of 3 cloud server facilities with different processing capacity capabilities and different PUE. We assume that the first cloud server facility is located in Ireland, the second in Spain and the third in Austria. Each cloud server facility is plugged to a wind turbine [19],[20] and [18]. Figure 28 shows the power generation of the three aforementioned wind turbines. Figure 29 shows how the online algorithm is able to execute more requests by selecting the appropriate cloud server facility and shifting the virtual machine execution.

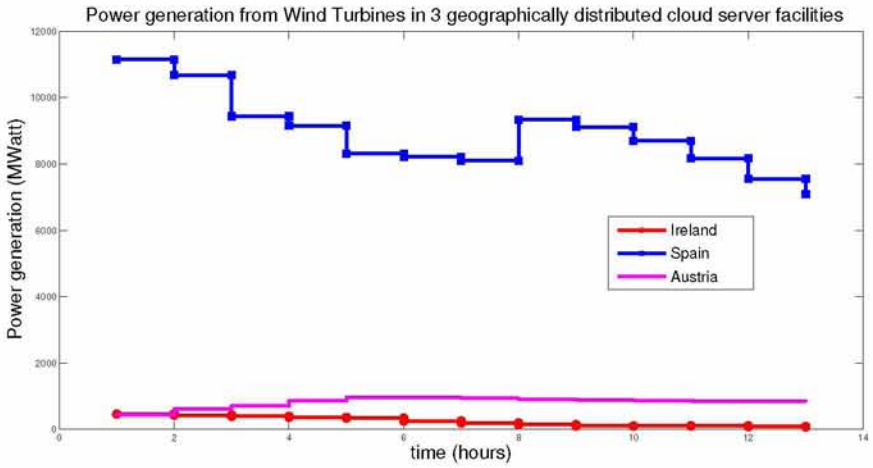

Figure 28: Power generation from wind turbines

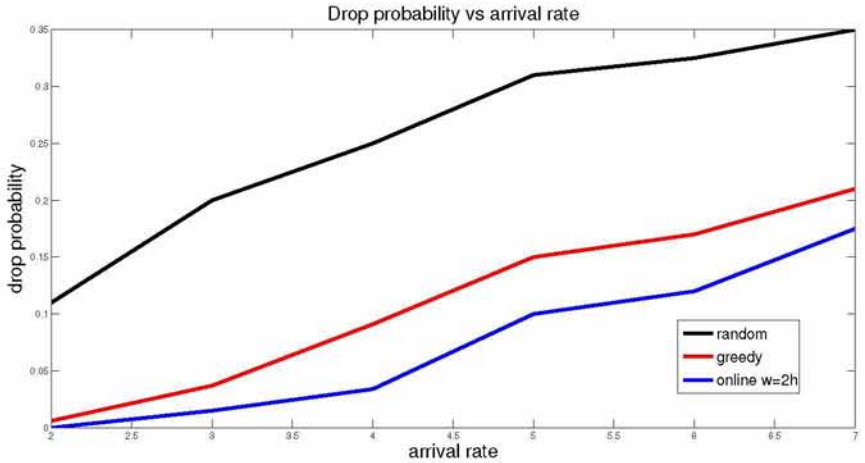

Figure 29: Drop probability increases as the arrival rate increases

11.3 Experiment Three

In order to test our algorithm we use real data traces. In case of wind turbine data we use [3] and for solar panels we use a Gaussian shape function with peak at midday. The peak of the Gaussian shape function of every solar RES is equal to 1.2 times the maximum power the cloud server which the RES supports can consume. In this experiment we assume that the rate for VM hosting dynamically changes per hour and we use as arrival rates the following normalized values [31]:

$$\lambda = [0.6\ 0.52\ 0.4\ 0.28\ 0.2\ 0.17\ 0.16\ 0.175$$
$$0.23\ 0.32\ 0.45\ 0.6\ 0.64\ 0.69\ 0.72\ 0.75$$
$$0.78\ 0.8\ 0.83\ 0.9\ 0.97\ 0.94\ 0.82\ 0.71]$$

where every value is the average arrival rate during one hour. The first value corresponds to 21.00 at night. The number of the flops and the deadline of every arriving request follow a uniform distribution.

The price of electricity from the power grid is the same to every data center and it is equal to 0.1265 euros per kWh. We create five geographically distributed data centers with a capacity of 1000,1500,800,1200,500 processing units respectively and we equip three of them with solar panels and two with wind turbines. Every data center is located in a different time zone (UTC 0,UTC -7,UTC 3 ,UTC 5,UTC 12) and that means that the solar panels are not at their peak at the same time.

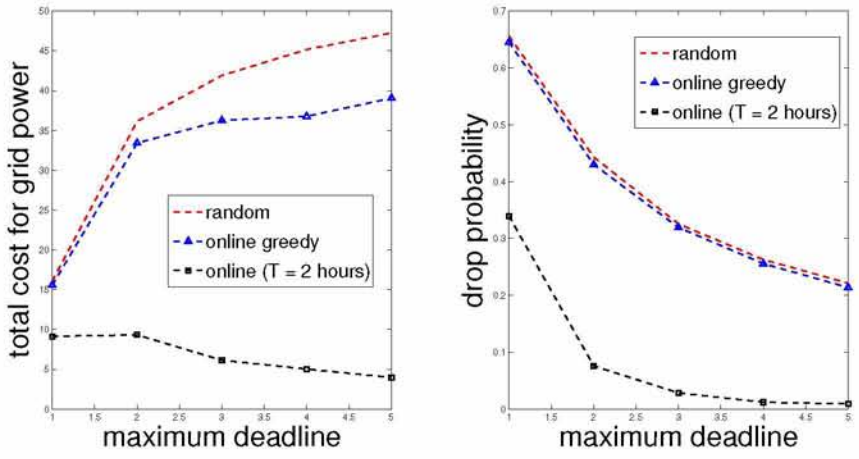

Figure 30: Total cost and drop probability vs maximum deadline.

Figure 30 shows how the deadline imposed in the request affects the operational cost of the cloud server facility and the drop probability. The

drop probability defines the case where the requested resources to one of the cloud servers can not be served. Random and greedy algorithms do not have the ability to postpone the hosting of the VM for the near future and that increases the probability of being unable to host the VM, if the deadline is too short. On the other hand the online algorithm utilizes the system resources more effectively because it can start the hosting of a new request after some resources will be released. This reduces the probability of dropping the request.

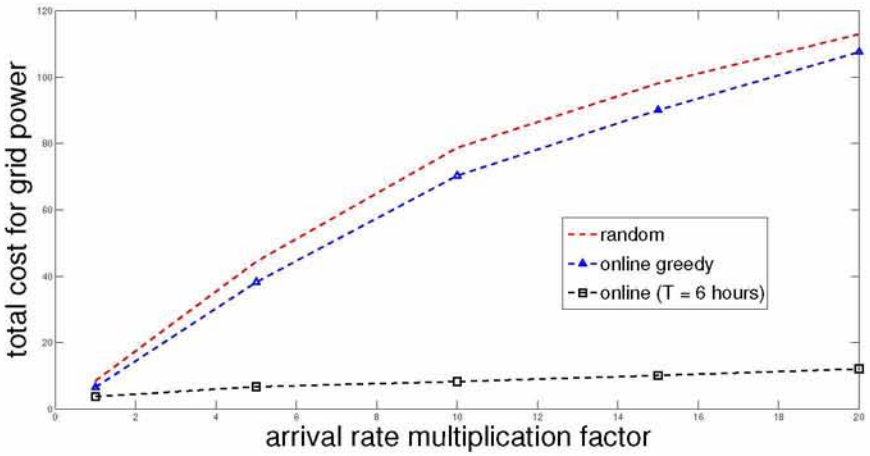

Figure 31: Total cost vs arrival rate multiplication factor.

In the same setting of five servers, in one day horizon, each arriving request imposes a deadline with a maximum value of 4 hours and an average number of 2000 flops. Figure 11.3 presents the average total cost of the cloud facility as a function of the arrival rate, over 50 independent simulation runs. The x-axis is the multiplier, by which we multiply λ. Online algorithm with lookahead window of 6 hours causes a reduction to power costs because it exploits the RES in a better way since it controls both the start of the execution of every job and its processing capacity.

Figure 32 depicts the reduction of the power cost as a function of the lookahead window. Furthermore because of the geographical distribution of the cloud servers, which results to the inability of providing a low cost service at the same time, the power grid cost increases when both the number of cloud facilities and the arrival rate multiplication factor are doubled.

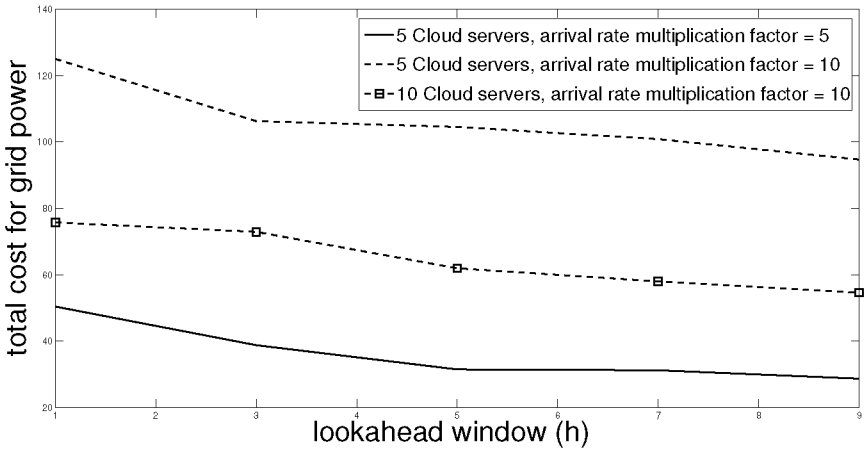

Figure 32: One day horizon. Cloud servers with equal capacity 5000 FLOPs/second, each arriving request imposes a deadline with a maximum value of 20 hours

11.4 Experiment Four

The setting of that experiment consists of four cloud servers with equal capacity of 5000 flops/second which are located in different time-zones and are supported by solar panels. Figure 11.4 depicts the power needs of every cloud server facility for the two online algorithms (blue and black lines) and the power generation of the renewable energy source which is plugged to every cloud server facility (green line). We note that the online algorithm with lookahead window of 6 hours selects the cloud server facility which offers less cost that the one with lookahead window of 2 hours. The arrival rates are 15 times the vector λ, the maximum deadline is 20 hours and the maximum number of flops is 1000.

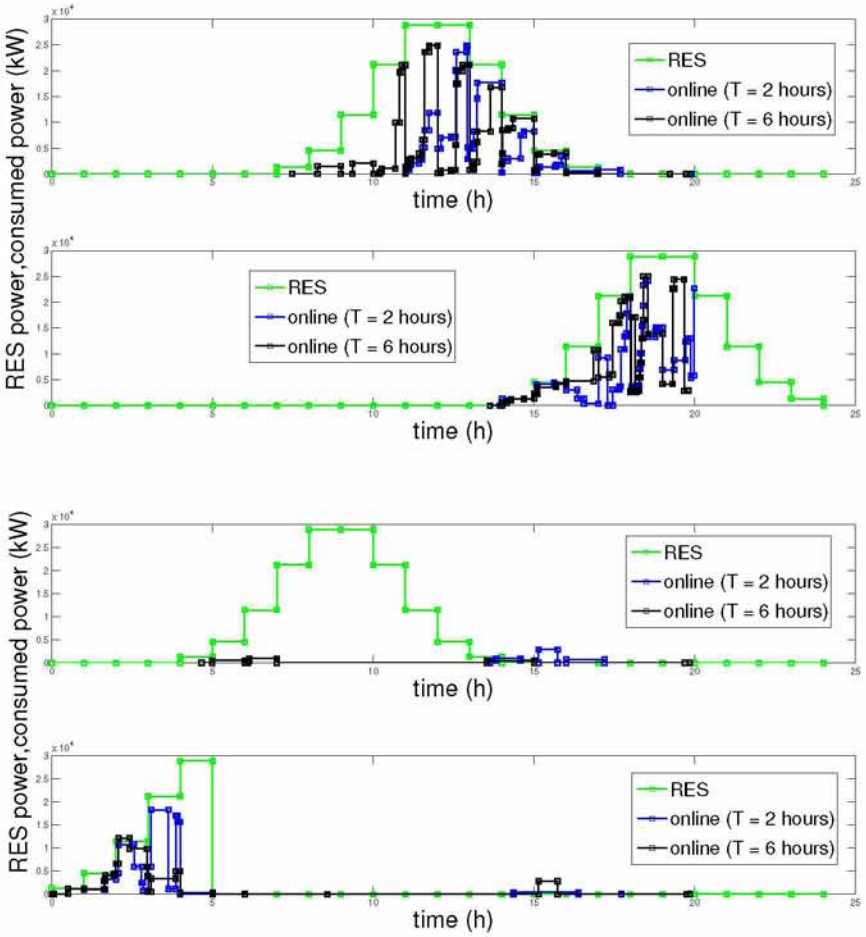

Figure 33: Power consumption and RES power supply (kW) during one day for 4 cloud facilities.

11.5 Experiment Five

In this experiment we depict the impact of the power consumption when the cloud server facility is idle (i.e P_{idle} of equation 2). We assume five georgaphically distributed cloud server facilities (australia,canada,hong kong,japan,south africa) with different price per power unit for the power from the power grid and also power supply from renewable energy sources. Figure 34 shows how the power consumption of the idle resources increases the cost.

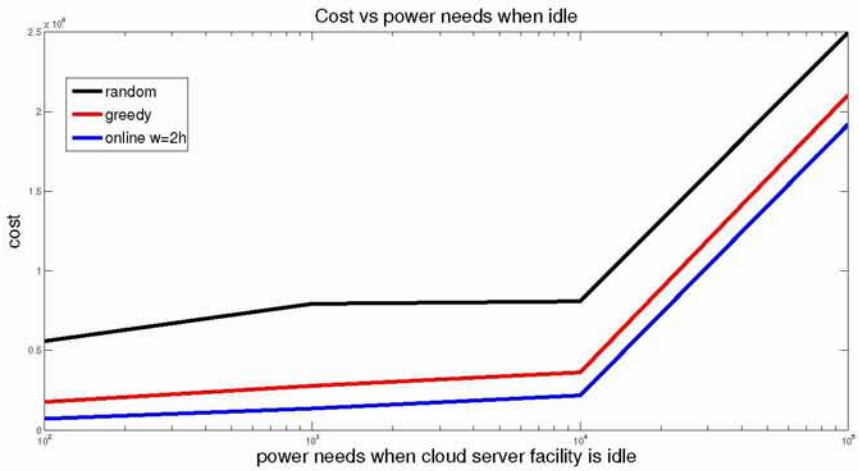

Figure 34: Total cost vs power consumption when the cloud server facility is idle. For every value of the arrival rate we conduct the same experiment 1000 times.

12 Pricing Algorithms

The previous sections are devoted to algorithms that minimize the cost of providing cloud computing service by a cloud provider. In that section we provide a pricing strategy for the cloud provider. More specifically, for every request j, cloud provider after the hosting decitions about the start execution time s_j, the running/hosting time τ_j, the processing capacity c_j and the cloud server facility h_j that will host the virtual machine which executes the request takes one more decition which is the price m_j the client will pay for that request.

Cloud providers want to sell their services at high prices for maximising their benefit. However, clients have possibility of election, and will choose the cheapest provider for the same service. The freedom of election of the client depends on the status of the demand for that cloud service. Cloud providers can raise their prices when the demand is high, and they must decrease prices when the demand is lower than the offer [37].

Given the fact that the client selected a specific cloud provider, the price he will pay for his request should be as little as possible since we assume that the client made a reasonable choice. On the other hand cloud provider could not charge for a request less money than the execution cost of that request.

The execution cost e_j of a specific request is:

$$e_j = \int_{s_j}^{s_j+\tau_j} p_{h_j}(t) \left[\left[f_{h_j}(L_{h_j}(t) + c_j) - R_{h_j}(t) \right]^+ - \left[f_{h_j}(L_{h_j}(t)) - R_{h_j}(t) \right]^+ \right] dt \tag{47}$$

In the case of an underloaded cloud server facility with enough power generation from the RES, e_j could be close to zero. Nonetheless m_j should be greater than zero.

A realistic strategy is to define a minimum price m_{min} which would be applied if $e_j < m_{min}$ and a minimum revenue m_r in the case of $e_j \geq m_{min}$ such that:

$$m_j = e_j + m_r \tag{48}$$

Factors like the popularity, the market share and the financial status of the cloud provider determine m_{min} and m_r.

If we assume that the client will pay for the execution after its end or at its deadline, then m_j can be calculated easily. Otherwise, cloud provider has to propose the value of m_j without knowing the exact value of e_j because he is unaware of the stochastic parameters of the system. A simple algorithm

who solves this problem is a learning algorithm who adjusts the parameters m_r and m_{max} in such a way to achive an average value of m_{min} and m_r in a long term.

13 Related Work

There has been a lot of work on VM allocation in cloud computing environments. However, only our work deals with the problem of VM allocation under a deadline constraint in a cloud computing environment with unpredictable power supply from RESs in such a way to minimize the total cost of the power supply from the power grid. Additionally, most of the work done in the area of VM allocation assumes distinct VM configurations.

The authors of [38] propose one preemptive and one non-preemptive throughput optimal scheduling algorithm. There are specific VM configurations. Requests for these configurations arrive at a central scheduler and are queued there. The scheduler dispatches a request to a server when the server has enough resources to host the requested VM. The work of [40] proposed a two time scale algorithm for power cost reduction in geographically distributed data centers. The control variables are the rooting distribution, the number of the servers that would be turned on in order to process the queued jobs and the processing rate for all the active servers in the cluster.

A different work is [32] which takes the challenge to reduce the electric bill of commercial web search engines operating on data centers that are geographically far apart. To achieve that, the authors propose a technique based on the observation that energy prices and query workloads show high spatiotemporal variation. The authors of [39] characterize the variation due to fluctuating electricity prices and argue that existing distributed systems should be able to exploit this variation for significant economic gains. Another idea to reduce power cost is to use RESs because of their low cost. The authors of [21] propose the adoption of accurate solar and wind energy predictors more efficient than state-of-the-art time series models.

In addition, the authors of [36] and [35] evaluate the impact of geographical load balancing, the optimal mix of RESs and the role of storage in order to investigate the feasibility of powering internet-scale systems using entirely renewable energy. Furthermore, the authors of [34] model the energy flows in a data center and optimize its holistic operation. They design an IT workload manager that schedules IT workload and allocates IT resources within a data center according to time varying power supply and cooling efficiency using RESs and IT demand predictions

14 Conclusion

We propose an online allocation algorithm with a lookahead window between which the RES capabilities and the price of the power drawn from the power grid are known. We compare that algorithm with a naive algorithm and a greedy online algorithm and we show that it has better performance in terms of total cost of the grid power and utilization of the system resources. Furthermore, we design a slotted offline algorithm who is aware of the power grid prices, the renewable power generation and the request arrivals for a given time horizon. That algorithm analyzes all the valid possible senaria of execution for the given set of requests and finds the execution with the minimum cost.

References

[1] Amazon relational database service. http://aws.amazon.com/rds/.

[2] Breadth-first search, wikipedia entry. https://en.wikipedia.org/wiki/Breadth-first_search.

[3] Centre for renewable energy sources and saving. http://www.cres.gr/kape/index_gr.htm.

[4] Cloud computing, wikipedia entry. http://en.wikipedia.org/wiki/Cloud_computing.

[5] Clouds 360. http://www.clouds360.com/.

[6] Depth-first search, wikipedia entry. http://en.wikipedia.org/wiki/Depth-first_search.

[7] Differentiation under the integral sign, wikipedia entry. http://en.wikipedia.org/wiki/Differentiation_under_the_integral_sign.

[8] Electricity pricing, wikipedia entry. http://en.wikipedia.org/wiki/Electricity_pricing.

[9] The facts on electricity prices. http://www.ret.gov.au/Department/Documents/clean-energy-future/ELECTRICITY-PRICES-FACTSHEET.pdf.

[10] Floating-point operations per second, wikipedia entry. http://en.wikipedia.org/wiki/FLOPS.

[11] Independent electricity system operator (ieso). http://www.ieso.ca/imoweb/marketdata/markettoday.asp.

[12] Service-level agreement, wikipedia entry. http://en.wikipedia.org/wiki/Service-level_agreement.

[13] Shortest path problem, wikipedia entry. http://en.wikipedia.org/wiki/Shortest_path_problem.

[14] Sorting algorithm, wikipedia entry. https://en.wikipedia.org/wiki/Sorting_algorithm.

[15] Virtual machine, wikipedia entry. http://en.wikipedia.org/wiki/Virtual_machine.

[16] Virtualization, wikipedia entry. http://en.wikipedia.org/wiki/Virtualization.

[17] What is cloud computing, amazon web services. http://aws.amazon.com/what-is-cloud-computing/.

[18] Wind generation, austrian power grid. http://www.apg.at/en/market/generation/wind-energy.

[19] Wind generation, eirgrid. http://www.eirgrid.com/operations/systemperformancedata/windgeneration/.

[20] Wind generation, red electrica de espana. https://demanda.ree.es/eolica.html.

[21] B. Aksanli, J. Venkatesh, L. Zhang, and T. Rosing. Utilizing green energy prediction to schedule mixed batch and service jobs in data centers. In *Proc. ACM HOTPOWER*, 2011.

[22] L.A. Barroso and U. Hlzle. *The data center as a computer: An introduction to the design of warehouse-scale machines*. Morgan and Claypool, 2009.

[23] Salman A. Baset. Cloud slas: Present and future. ACM Operating System Review, 2012.

[24] M. Blackburn. *Five ways to reduce data center server power consumption*. The Green Grid, White Paper, 2008.

[25] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Wareld. Live migration of virtual machines. In *Proc. USENIX*, 2005.

[26] Aake Edlund. Pay-as-you-go computing explained. 2009.

[27] Lakshmi Ganesh. *DATA CENTER ENERGY MANAGEMENT*. PhD thesis, Cornell University, 2012.

[28] Dimitrios Hatzopoulos, Iordanis Koutsopoulos, George Koutitas, and Ward Van Heddeghem. Dynamic virtual machine allocation in cloud server facility systems with renewable energy sources. In *IEEE international conference on Communications*, 2013.

[29] Jianhui Huang. *PRICING STRATEGY FOR CLOUD COMPUTING SERVICES*. PhD thesis, School of Information Systems, Singapore Management University,Singaopre, 2013.

[30] Fredrich Kahrl, Jim Williams, and Ding Jianhua. Four things you should know about chinas electricity system. Chine Environment forum.

[31] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken. The nature of datacenter traffic: Measurements and analysis. In *Proc. ACM IMC*, 2009.

[32] E. Kayaaslan, B.B. Cambazoglu, R. Blanco, F.P. Junqueira, and C. Aykanat. Energy price-driven query processing in multi-center web search engines. In *Proc. ACM SIGIR*, 2011.

[33] G. Koutitas and P. Demestichas. Challenges for energy efficiency in local and regional datacenters. In *Journal Green Engineering, vol. 1, no.1,*, 2010.

[34] Z. Liu, Y. Chen, C. Bash, A. Wierman, D. Gmach, Z. Wang, M. Marwah, and C. Hyser. Renewable and cooling aware workload management for sustainable data centers. In *Proc. ACM SIGMETRICS*, 2012.

[35] Z. Liu, M. Lin, A. Wierman, S.H. Low, and L.L. H. Andrew. Greening geographical load balancing. In *Proc. ACM SIGMETRICS*, 2011.

[36] Z. Liu, M. Lin, A. Wierman, S.H. Low, and L.L.H Andrew. Geographical load balancing with renewables. In *Proc. ACM SIGMETRICS*, 2011.

[37] Mario Macas and Jordi Guitart. A genetic model for pricing in cloud computing markets. In *ACM Symposium on Applied Computing*, pages 113–118, 2011.

[38] S.T Maguluri, R. Srikant, and L. Ying. Stochastic models of load balancing and scheduling in cloud computing clusters. In *Proc. IEEE INFOCOM*, 2012.

[39] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs. Cutting the electric bill for internet-scale systems. In *Proc. ACM SIGCOMM*, 2009.

[40] Y. Yao, L. Huang, A. Sharma, L. Golubchik, and M. J. Neely. Data centers power reduction: A two time scale approach for delay tolerant workloads. In *Proc. IEEE INFOCOM*, 2012.