This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2396059, IEEE Transactions on Cloud Computing

Energy-aware Load Balancing and Application Scaling for the Cloud Ecosystem

Ashkan Paya and Dan C. Marinescu

Computer Science Division, EECS Department
University of Central Florida, Orlando, FL 32816, USA
Email:ashkan_paya@knights.ucf.edu, dcm@cs.ucf.edu

Abstract - In this paper we introduce an energy-aware operation model used for load balancing and application scaling on a cloud. The basic philosophy of our approach is defining an energy-optimal operation regime and attempting to maximize the number of servers operating in this regime. Idle and lightly-loaded servers are switched to one of the sleep states to save energy. The load balancing and scaling algorithms also exploit some of the most desirable features of server consolidation mechanisms discussed in the literature.

Index terms - load balancing, application scaling, idle servers, server consolidation, energy proportional systems.

1 Motivation and Related Work

In the last few years packaging computing cycles and storage and offering them as a metered service became a reality. Large farms of computing and storage platforms have been assembled and a fair number of Cloud Service Providers (CSPs) offer computing services based on three cloud delivery models SaaS (Software as a Service), PaaS (Platform as a Service), and IaaS (Infrastructure as a Service).

Warehouse-scale computers (WSCs) are the building blocks of a cloud infrastructure. A hierarchy of networks connect $50,000$ to $100,000$ servers in a WSC. The servers are housed in racks; typically, the 48 servers in a rack are connected by a 48-port Gigabit Ethernet switch. The switch has two to eight up-links which go to the higher level switches in the network hierarchy [4, 17].

Cloud elasticity, the ability to use as many resources as needed at any given time, and low cost, a user is charged only for the resources it consumes, represents solid incentives for many organizations to migrate their computational activities to a public cloud. The number of CSPs, the spectrum of services offered by the CSPs, and the number of cloud users have increased dramatically during the last few years. For example, in 2007 the EC2 (Elastic Computing Cloud) was the first service provided by AWS (Amazon Web Services); five years later, in 2012, AWS was used by businesses in 200 countries. Amazon's S3 (Simple Storage Service) has surpassed two trillion objects and routinely runs more than 1.1 million peak requests per second. Elastic MapReduce has launched 5.5 million clusters since May 2010 when the service started [35].

The rapid expansion of the cloud computing has a significant impact on the energy consumption in US and the world. The costs for energy and for cooling large-scale data centers are significant and are expected to increase in the future. In 2006, the 6 000 data centers in the U.S. reportedly consumed 61×10^9 kWh of energy, 1.5% of all electricity consumption in the country, at a cost of $4.5 billion [34]. The energy consumption of data centers and of the network infrastructure is predicted to reach $10,300$ TWh/year (1 TWh = 10^9 kWh) in 2030, based on 2010 efficiency levels [28]. These increases are expected in spite of the extraordinary reduction in energy requirements for computing activities.

Idle and under-utilized servers contribute significantly to wasted energy, see Section 2. A 2010 survey [8] reports that idle servers contribute 11 million tons of unnecessary CO_2 emissions each year and that the total yearly costs for idle servers is $19 billion. Recently, Gartner Research [30] reported that the average server utilization in large data-centers is 18%, while the utilization of *x86* servers is even lower, 12%. These results confirm earlier estimations that the average server utilization is in the $10 - 30\%$ range [3].

The concept of "load balancing" dates back to the time when the first distributed computing systems were implemented. It means exactly what the name implies, *to evenly distribute the workload to a set of servers* to maximize the throughput, minimize the response time, and increase the system resilience to faults by avoiding overloading the systems.

An important strategy for energy reduction is concentrating the load on a subset of servers and, whenever possible, switching the rest of them to a state with a low energy consumption. This observation implies that the traditional concept of load balancing in a large-scale system could be reformulated as follows: *distribute evenly the workload to the smallest set of servers operating at optimal or near-optimal energy levels, while observing the Service Level Agreement (SLA) between the CSP and a cloud user. An optimal energy level is one when the performance per Watt of power is maximized.*

Scaling is the process of allocating additional resources to a cloud application in response to a request consistent with the SLA. We distinguish two scaling modes, horizontal and vertical scaling. *Horizontal scaling* is the most common mode of scaling on a cloud; it is done by increasing the number of Virtual Machines (VMs) when the load of applications increases and reducing this number when the load decreases.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2396059, IEEE Transactions on Cloud Computing

Load balancing is critical for this mode of operation. *Vertical scaling* keeps the number of VMs of an application constant, but increases the amount of resources allocated to each one of them. This can be done either by migrating the VMs to more powerful servers or by keeping the VMs on the same servers, but increasing their share of the server capacity. The first alternative involves additional overhead; the VM is stopped, a snapshot is taken, the file is migrated to a more powerful server, and the VM is restarted at the new site.

The alternative to the wasteful resource management policy when the servers are *always on*, regardless of their load, is to develop *energy-aware load balancing and scaling* policies. Such policies combine *dynamic power management* with load balancing and attempt to identify servers operating outside their optimal energy regime and decide if and when they should be switched to a sleep state or what other actions should be taken to optimize the energy consumption. The vast literature on energy-aware resource management concepts and ideas discussed in this paper includes [1, 3, 5, 6, 7, 10, 11, 22, 25, 28, 32, 33, 34].

Some of the questions posed by energy-aware load balancing and application scaling are: (a) Under what conditions should a server be switched to a sleep state? (b) What sleep state should the server be switched to? (c) How much energy is necessary to switch a server to a sleep state and then switch it back to an active state? (d) How much time it takes to switch a server to a running state from a sleep state? (e) How much energy is necessary for migrating a VM running on a server to another one? (f) How much energy is necessary for starting the VM on the target server? (g) How to choose the target where the VM should migrate to? (h) How much time does it takes to migrate a VM?

The answers to some of these questions depend on the server's hardware and software, including the virtual machine monitor and the operating systems, and change as the technology evolves and energy awareness becomes increasingly more important. In this paper we are concerned with high-level policies which, to some extent are independent of the specific attributes of the server's hardware and, due to space limitation, we only discuss (a), (b), and (g). We assume that the workload is predictable, has no spikes, and that the demand of an application for additional computing power during an evaluation cycle is limited. We also assume a *clustered organization*, typical for existing cloud infrastructure [4, 17].

There are three primary contributions of this paper: (1) a new model of cloud servers that is based on different operating regimes with various degrees of "energy efficiency" (processing power versus energy consumption); (2) a novel algorithm that performs load balancing and application scaling to maximize the number of servers operating in the energy-optimal regime; and (3) analysis and comparison of techniques for load balancing and application scaling using three differently-sized clusters and two different average load profiles.

Models for energy-aware resource management and application placement policies and the mechanisms to enforce these policies such as the ones introduced in this paper can be evaluated theoretically [1], experimentally [10, 11, 13, 22], through simulation [5, 7, 27], based on published data [2, 8, 19, 20], or through a combination of these techniques. Analytical models can be used to derive high-level insight on the behavior of the system in a very short time but the biggest challenge is in determining the values of the parameters; while the results from an analytical model can give a good approximation of the relative trends to expect, there may be significant errors in the absolute predictions. Experimental data is collected on small-scale systems; such experiments provide useful performance data for individual system components but no insights on the interaction between the system and applications and the scalability of the policies. Trace-based workload analysis such as the ones in [10] and [33] are very useful though they provide information for a particular experimental set-up, hardware configuration, and applications. Typically trace-based simulation need more time to produce results. Traces can also be very large and it is hard to generate representative traces from one class of machines that will be valid for all the classes of simulated machines. To evaluate the energy aware load balancing and application scaling policies and mechanisms introduced in this paper we chose simulation using data published in the literature [4].

Operating efficiency of a system and server consolidation are discussed in Sections 2 and 3, respectively. The model described in Section 4 introduces the operating regimes of a processors and the conditions when to switch a server to a sleep state. Load balancing and scaling algorithms suitable for a clustered cloud organization based on the model are presented in Section 5; these algorithms aim to optimize the energy efficiency and to balance the load. Simulations experiments and conclusions are covered in Sections 6 and 7.

2 Energy Efficiency of a System

The *energy efficiency* of a system is captured by the ratio "performance per Watt of power." During the last two decades the performance of computing systems has increased much faster than their energy efficiency [3].

Energy proportional systems. In an ideal world, the energy consumed by an idle system should be near zero and grow linearly with the system load. In real life, even systems whose energy requirements scale linearly, when idle, use more than half the energy they use at full load. Data collected over a long period of time shows that the typical operating regime for data center servers is far from an optimal energy consumption regime.

An *energy-proportional* system consumes no energy when idle, very little energy under a light load, and gradually, more energy as the load increases. An ideal energy-proportional system is always operating at 100% efficiency [3].

Energy efficiency of a data center; the dynamic range of subsystems. The energy efficiency of a data center is measured by the *power usage effectiveness* (PUE), the ratio of total energy used to power a data center to the energy used to power computational servers, storage servers, and other IT equipment. The PUE has improved from around 1.93 in 2003 to 1.63 in 2005; recently, Google reported a PUE ratio as low

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2396059, IEEE Transactions on Cloud Computing

as 1.15 [2]. The improvement in PUE forces us to concentrate on energy efficiency of computational resources [7].

The *dynamic range* is the difference between the upper and the lower limits of the energy consumption of a system as a function of the load placed on the system. A large dynamic range means that a system is able to operate at a lower fraction of its peak energy when its load is low.

Different subsystems of a computing system behave differently in terms of energy efficiency; while many processors have reasonably good energy-proportional profiles, significant improvements in memory and disk subsystems are necessary. The largest consumer of energy in a server is the processor, followed by memory, and storage systems. Estimated distribution of the peak power of different hardware systems in one of the Google's datacenters is: CPU 33%, DRAM 30%, Disks 10%, Network 5%, and others 22% [4] .

The power consumption can vary from 45W to 200W per multi-core CPU. The power consumption of servers has increased over time; during the period 2001 - 2005 the estimated average power use has increased from 193 to 225 W for volume servers, from 457 to 675 for mid range servers, and from 5,832 to 8,163 W for high end ones [19]. Volume servers have a price less than $25 K, mid-range servers have a price between $25 K and $499 K, and high-end servers have a price tag larger than $500 K. Newer processors include power saving technologies.

The processors used in servers consume less than one-third of their peak power at very-low load and have a dynamic range of more than 70% of peak power; the processors used in mobile and/or embedded applications are better in this respect. According to [3], the dynamic power range of other components of a system is much narrower: less than 50% for DRAM, 25% for disk drives, and 15% for networking switches. Large servers often use $32-64$ dual in-line memory modules (DIMMs); the power consumption of one DIMM is in the 5 to 21 W range.

A server with $2-4$ hard disk drives (HDDs) consumes $24-48$ W. A strategy to reduce energy consumption by disk drives is concentrating the workload on a small number of disks and allowing the others to operate in a low-power mode. One of the techniques to accomplish this is based on replication [34]. Another technique is based on data migration [15].

A number of proposals have emerged for *energy proportional* networks; the energy consumed by such networks is proportional with the communication load. An example of an energy proportional network is the *InfiniBand*. A network-wide power manager, which dynamically adjusts the network links and switches to satisfy changing datacenter traffic loads, called Elastic Tree is described in [16].

Sleep states. The effectiveness of sleep states in optimizing energy consumption is analyzed in [11]. A comprehensive document [18] describes the advanced configuration and power interface (ACPI) specifications which allow an operating system (OS) to effectively manage the power consumption of the hardware.

Several types of *sleep sates*, are defined: C-states ($C1-C6$) for the CPU, D-states ($D0-D3$) for modems, hard-drives, and CD-ROM, and S-states ($S1-S4$) for the basic input-output system (BIOS). The C-states allow a computer to save energy when the CPU is idle. In a sleep state, the idle units of a CPU have their clock signal and the power cut. The higher the state number, the deeper the CPU sleep mode, the larger the energy saved, and the longer the time for the CPU to return to the state $C0$ which corresponds to the CPU fully operational. The clock signal and the power of different CPU units are cut in states $C1$ to $C3$, while in state $C4$ to $C6$ the CPU voltage is reduced. In state $C1$ the main internal CPU clock is stopped by the software, but the bus interface and the advanced programmable interrupt controller (APIC) are running, while in state $C3$ all internal clocks are stopped, and in state $C4$ the CPU voltage is reduced.

Resource management policies for large-scale data centers. These policies can be loosely grouped into five classes: (a) Admission control; (b) Capacity allocation; (c) Load balancing; (d) Energy optimization; and (e) Quality of service (QoS) guarantees. The explicit goal of an admission control policy is to prevent the system from accepting workload in violation of high-level system policies; a system should not accept additional workload preventing it from completing work already in progress or contracted. Limiting the workload requires some knowledge of the global state of the system; in a dynamic system such knowledge, when available, is at best obsolete. Capacity allocation means allocating resources for individual instances; an instance is an activation of a service. Some of the mechanisms for capacity allocation are based on either static or dynamic thresholds [23].

Economy of scale affects the energy efficiency of data processing. For example, Google reports that the annual energy consumption for an Email service varies significantly depending on the business size and can be 15 times larger for a small business than for a large one [13]. Cloud computing can be more energy efficient than on-premise computing for many organizations [2, 26].

3 Server Consolidation

The term *server consolidation* is used to describe: (1) switching idle and lightly loaded systems [33] to a sleep state; (2) workload migration to prevent overloading of systems [7]; or (3) any optimization of cloud performance and energy efficiency by redistributing the workload [25].

Server consolidation policies. Several policies have been proposed to decide when to switch a server to a sleep state. The *reactive* policy [31] responds to the current load; it switches the servers to a sleep state when the load decreases and switches them to the running state when the load increases. Generally, this policy leads to SLA violations and could work only for slow-varying, predictable loads. To reduce SLA violations one can envision a *reactive with extra capacity* policy when one attempts to have a safety margin and keep running a fraction of the total number of servers, e.g., 20% above those needed for the current load. The *AutoScale* policy [10] is a very conservative *reactive* policy in switching servers to sleep state to avoid the power consumption and the delay in switching them back to running state. This can be

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2396059, IEEE Transactions on Cloud Computing

advantageous for unpredictable spiky loads.

A very different approach is taken by two versions of *predictive* policies [33]. The *moving window policy* estimates the workload by measuring the average request rate in a window of size Δ seconds uses this average to predict the load during the next second (second $(\Delta + 1)$) and then slides the window one second to predict the load for second $(\Delta + 2)$, and so on. The *predictive linear regression* policy uses a linear regression to predict the future load.

Optimal policy. In this paper we define an *optimal* policy as one which guarantees that running servers operate within their optimal energy regime or for limited time in a suboptimal regime, and, at the same time, the policy does not produce any SLA violations. At this time SLAs are very general and do not support strict QoS guarantees, e.g., hard real-time deadlines. Different policies can be ranked by comparing them with an *optimal* policy. The mechanisms to implement energy-aware application scaling and load balancing policies should satisfy several conditions: (i) Scalability - work well for large farms of servers; (ii) Effectiveness - lead to substantial energy and cost savings; (iii) Practicality - use efficient algorithms and require as input only data that can be measured with low overhead and, at the same time, accurately reflects the state of the system; and last, but not least, (iv) Consistency with the global objectives and contractual obligations specified by Service Level Agreements.

The workload can be slow- or fast-varying, have spikes or be smooth, can be predicted or is totally unpredictable; the admission control can restrict the acceptance of additional load when the available capacity of the servers is low. What we can measure in practice is the *average energy* used and the average server *setup time*. The setup time varies depending on the hardware and the operating system and can be as large as 260 seconds; the energy consumption during the setup phase is close to the maximal one for the server [10].

The time to switch the servers to a running state is critical when the load is fast varying, the load variations are very steep, and the spikes are unpredictable. The decisions when to switch servers to a sleep state and back to a running state are less critical when a strict admission control policy is in place; then new service requests for large amounts of resources can be delayed until the system is able to turn on a number of sleeping servers to satisfy the additional demand.

4 The System Model

The model introduced in this section assumes a clustered organization of the cloud infrastructure and targets primarily the IaaS cloud delivery model represented by Amazon Web Services (AWS). AWS supports a limited number of instance families, including M3 (general purpose), C3 (compute optimized), R3 (memory optimized), I2 (storage optimized), G2 (GPU) and so on. An *instance* is a package of system resources; for example, the `c3.8xlarge` instance provides 32 vCPU, 60 GiB of memory, and 2×320 GB of SSD storage. AWS used to measure the server performance in ECUs (Elastic Compute Units) but has switched recently to, yet to be specified, vCPU units; one ECU is the equivalent CPU capacity of a $1.0 - 1.2$ GHz 2007 Opteron or 2007 Xeon processor.

Applications for one instance family have similar profiles, e.g., are CPU-, memory-, or I/O-intensive and run on clusters optimized for that profile; thus, the application interference with one another is minimized. The normalized system performance and the normalized power consumption are different from server to server; yet, warehouse scale computers supporting an instance family use the same processor or family of processors [4] and this reduces the effort to determine the parameters required by our model.

In our model the migration decisions are based solely on the vCPU units demanded by an application and the available capacity of the host and of the other servers in the cluster. The model could be extended to take into account not only the processing power, but also the dominant resource for a particular instance family, e.g., memory for R3, storage for I2, GPU for G2 when deciding to migrate a VM. This extension would complicate the model and add additional overhead for monitoring the application behavior.

The model defines an energy-optimal regime for server operation and the conditions when a server should be switched to a sleep state. Also the model gives some hints regarding the most appropriate sleep state the server should be switched to and supports a decision making framework for VM migration in horizontal scaling.

Clustered organization. A cluster $\mathcal{C}$ has a *leader*, a system which maintains relatively accurate information about the free capacity of individual servers in the cluster and communicates with the leaders of the other clusters for the implementation of global resource management policies. The leader could consist of a multi-system configuration to guarantee a fast response time and to support fault-tolerance.

An advantage of a clustered organization is that a large percentage of scheduling decisions are based on local, therefore more accurate, information. Server $\mathcal{S}_k$ makes scheduling decisions every τ_k units of time. The servers in the cluster report to the leader the current load and other relevant state information every τ^i units of time, or earlier if the need to migrate an application is anticipated.

We consider three levels of resource allocation decision making: (a) the local system which has accurate information about its state; (b) the cluster leader which has less accurate information about the servers in the cluster; and (c) global decisions involving multiple clusters. In this paper we are only concerned with *in-cluster scheduling* coordinated by the leader of the cluster. *Inter-cluster scheduling* is based on less accurate information as the leader of cluster $\mathcal{C}$ exchanges information with other leaders less frequently.

System and application level resource management. The model is based on a two-level decision making process, one at the system and one at the application level. The scheduler of the *Virtual Machine Monitor* (VMM) of a server interacts with the *Server Application Manager (SAM)* component of the VMM to ensure that the QoS requirements of the application are satisfied. SAM gathers information from individual application managers of the VMs running on the server.

Model parameters. The cluster leader maintains static

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2396059, IEEE Transactions on Cloud Computing

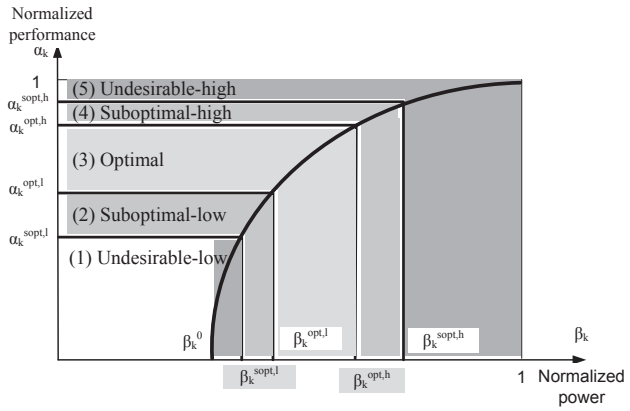


Figure 1: Normalized performance versus normalized power; boundaries of the five operating regimes are shown.

and dynamic information about all servers in cluster $\mathcal{C}$. *Static information* includes:

$\mathcal{S}_k$ - the serverId;

γ_k - a constant quantifying the processing power of server $\mathcal{S}_k$; the processing power is expressed in vCPUs.

The model illustrated in Figure 1 uses several parameters: $\alpha_k^{sopt,l}$ $\alpha_k^{opt,l}$, $\alpha_k^{opt,h}$, and $\alpha_k^{sopt,h}$, the normalized performance boundaries of different operating regimes.

τ_k - the reallocation interval. Ever τ_k units of time the system determines if and how to reallocate resources.

The application record of application $\mathcal{A}_{i,k}$ includes the *applicationId* and several other parameters:

(1) $a_{i,k}(t)$ - current demand of application $\mathcal{A}_i$ for processing power on server $\mathcal{S}_k$ at time t in vCPU units, e.g., 0.4.

(2) $\lambda_{i,k}$ - highest rate of increase in demand for processing power of application $\mathcal{A}_i$ on server $\mathcal{S}_k$.

(3) $p_{i,k}(t)$ - migration cost.

(4) $q_{i,k}(t)$ - horizontal scaling cost.

The increase in demand for processing power of application $\mathcal{A}_{i,k}$ during a reallocation interval is limited

$$a_{i,k}(t+\tau_k) \leq a_{i,k}(t) + \lambda_{i,k}\tau_k. \quad (1)$$

$a_k(t)$, the fraction of processing power of server $\mathcal{S}_k$ used by all active applications at time t is an example of *dynamic information*. This information is reported with period τ^i, or whenever the server determines that it needs to migrate an application or to create additional VMs for an application. To minimize communication costs, the reporting period τ^i is much larger than the rescheduling period of individual clusters. A server $\mathcal{S}_k$ has a computational constant γ_k which quantifies the highest level of performance it can deliver.

Server operating regimes. The normalized performance of server $\mathcal{S}_k$ depends on the power level, $\alpha_k(t) = f_k[\beta_k(t)]$. We distinguish five operating regimes of a server, an optimal one, two suboptimal, and two undesirable, Figure 1:

$\mathcal{R}_1$ - undesirable-low regime

$$\beta_k^0 \leq \beta_k(t) \leq \beta_k^{sopt,l} \quad \text{and} \quad 0 \leq a_k(t) \leq \alpha_k^{sopt,l}. \quad (2)$$

$\mathcal{R}_2, \mathcal{R}_3, \mathcal{R}_4$, - suboptimal-low, optimal, and suboptimal high regimes

$$\beta^{r1,llim} \leq \beta_k \leq \beta^{r2,hlim} \quad \text{and} \quad \alpha^{r1,llim} \leq \alpha_k \leq \alpha^{r2,hlim} \quad (3)$$

with:
$\mathcal{R}_2$: r1=sopt, r2=opt, llim=l, hlim=1
$\mathcal{R}_3$: r1=r2=opt, llim=l, hlim=h
$\mathcal{R}_4$: r1=opt, r2=sopt, llim=h, hlim=h

$\mathcal{R}_5$ - undesirable-high regime

$$\beta_k^{sopt,h} \leq \beta_k(t) \leq 1 \quad \text{and} \quad \alpha_k^{sopt,h} \leq a_k(t) \leq 1. \quad (4)$$

Our choice of five operating regimes is motivated by the desire to distinguish three types of system behavior in terms of power utilization: optimal, suboptimal, and undesirable. The optimal regime is power efficient and gives a degree of confidence that the sever will not be forced to request VM migrations during the next few scheduling cycles while the sub-optimal regimes are acceptable only for limited periods of time. The five-regime model coupled with delayed actions, e.g., allowing a system to operate within a suboptimal or undesirable-high regimes, could reduce the system overhead and even save power by reducing VM migration costs. One can consider a more refined system behavior, e.g., a seven-regime model, but this adds to the model complexity without clear benefits.

This classification captures the current system load and allows us to distinguish the actions to be taken to return to the optimal regime. A system operating in the suboptimal-low regime is lightly loaded; the server is a candidate for switching to a sleep state. The undesirable-high regime should be avoided because a scaling request would immediately trigger VM migration and, depending on the system load, would require activating one of the servers in a sleep state. This classification also captures the urgency of the actions taken; suboptimal regimes do not require an immediate attention, while the undesirable-low does. The time spent operating in each suboptimal regime is also important.

Measuring server energy efficiency. A recent benchmark [29] compares the energy efficiency of typical business applications running on a Java platform. For example, Table 1 based on data reported in Figure 5.3 of [4] shows the results for the SPECpower_ssj2008 benchmark for a server with a single chip 2.83 GHz quad core Intel Xeon processor, 4GB of DRAM, and one 7.2 k RPM 3.5" SATA disk drive.

From Table 1 we see that the energy efficiency is nearly-linear. Consider the case when the workload of n servers operating in the $\mathcal{R}_1$ regime migrates to n^{opt} servers already in the $\mathcal{R}_3$ regime and the n servers are forced to a sleep state. The energy saved over an interval of time T, $E^s(T)$, while delivering the same level of performance satisfies the condition

$$E^s(T) \geq n \times \beta_0 \times T - n \times (\bar{p} + \bar{s}) - n^{opt} \times (\beta^{opt,h} - \beta^{opt,l}), \quad (5)$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2396059, IEEE Transactions on Cloud Computing

Table 1: The average workload as a percentage of the maximal workload, the power used in Watts, the number of transactions, and the computational efficiency, the ratio of transactions to the average power consumption, from [4]

Load (%)	0	10	20	30	35	40	50	60	70	80	90	100
P - power (W)	165	180	185	190	195	200	210	220	225	235	240	250
T- transactions	0	175	335	484	552	620	738	854	951	1,049	1,135	1,214
Efficiency= T/P	0	0.97	1.81	2.55	2.83	3.10	3.51	3.88	4.23	4.46	4.73	4.84

assuming that all servers have the same: (1) energy consumption when idle, β_0; (2) average migration costs $\bar{p}$; and (3) average setup cost $\bar{s}$. The first term of Equation 5 shows that the longer the system operates in this mode, the larger the energy savings. The second term measures the energy used for VM migration and for server setup when they need to be brought back from the sleep state. The last term of Equation 5 accounts for the increase in energy consumption due to additional load of the n^{opt} servers operating in the optimal regime. For the example in Table 1, $\beta_0 = 165$ W. In [10] the setup time is 260 seconds and during that time the power is consumed at the peak rate of $\bar{s} = 200$ W. There are no reliable estimations of the migration costs. Processors with low-power halt states, such as the C1E state of `x86`, have a lower setup cost [4].

The results in Table 1 suggest the following boundaries for the five operating regions:

$$\alpha^{sub,l} = 35\%, \alpha^{opt,l} = 50\%, \alpha^{sub,l} = 80\%, \alpha^{und,h} = 90\% \quad (6)$$

These boundaries will be used for the simulation discussed in Section 6. The parameters for the five operating regimes $\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3, \mathcal{R}_4$ and $\mathcal{R}_5$ corresponding to the boundaries in Equation 6 are summarized in Table 2.

Table 2: The thresholds α, the average power consumption per processor, $\bar{P}$, the average performance measured as the number of transactions, $\bar{T}$, and the ratio $\bar{T}/\bar{P}$ for the five regimes: $\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3, \mathcal{R}_4$ and $\mathcal{R}_5$.

	$\mathcal{R}_1$	$\mathcal{R}_2$	$\mathcal{R}_3$	$\mathcal{R}_4$	$\mathcal{R}_5$
α	0-34.9	35-49.9	50-79.9	80-89.9	90-100
$\bar{P}$	180	202.5	222.5	237.5	245
$\bar{T}$	276	645	894	1,092	1,175
$\bar{T}/\bar{P}$	1.410	3.165	3.985	4.595	4.785

Application scaling. We assume that the SAM of server $\mathcal{S}_k$ updates every τ_k units of time the $a_{i,k}(t)$ of all applications $\mathcal{A}_{i,k}$ and predicts the consumption at the beginning of the next reallocation interval. The SAM maintains a control data structure including all currently running applications, ordered by $a_{i,k}(t)$. At each reallocation instance, server $\mathcal{S}_k$ determines its available capacity

$$d_k(t) = \alpha_k^{opt,h} - \frac{1}{\gamma_k} \sum_i a_{i,k}(t), \quad (7)$$

as well as the largest possible demand for processing capacity at the end of that reallocation interval

$$g_k(t + \tau_k) = \sum_i \left(a_{i,k}(t) + \lambda_{i,k}\tau_k\right). \quad (8)$$

5 Energy-aware Scaling Algorithms

The objective of the algorithms is to ensure that the largest possible number of active servers operate within the boundaries of their respective optimal operating regime. The actions implementing this policy are: (a) migrate VMs from a server operating in the undesirable-low regime and then switch the server to a sleep state; (b) switch an idle server to a sleep state and reactivate servers in a sleep state when the cluster load increases; (c) migrate the VMs from an overloaded server, a server operating in the undesirable-high regime with applications predicted to increase their demands for computing in the next reallocation cycles.

The clustered organization allows us to accommodate some of the desirable features of the strategies for server consolidation discussed in Section 3. For example, when deciding to migrate some of the VMs running on a server or to switch a server to a sleep state, we can adopt a conservative policy similar to the one advocated by autoscaling [10] to save energy. Predictive policies, such as the ones discussed in [33] will be used to allow a server to operate in a suboptimal regime when historical data regarding its workload indicates that it is likely to return to the optimal regime in the near future.

The cluster leader has relatively accurate information about the cluster load and its trends. The leader could use predictive algorithms to initiate a gradual wake-up process for servers in a deeper sleep state, $C4 - C6$, when the workload is above a "high water mark" and the workload is continually increasing. We set up the high water mark at 80% of the capacity of active servers; a threshold of 85% is used for deciding that a server is overloaded in [12], based on an analysis of workload traces. The leader could also choose to keep a number of servers in $C1$ or $C2$ states because it takes less energy and time to return to the $C0$ state from these states. The energy management component of the hypervisor can use only local information to determine the regime of a server.

Scaling decisions. The Server Application Manager SAM_k is a component of the Virtual Machine Monitor (VMM) of a server $\mathcal{S}_k$. One of its functions is to classify the applications based on their processing power needs over a window of w reallocation intervals in several categories: rapidly increasing resource demands (RI), moderately increasing (MI),

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2396059, IEEE Transactions on Cloud Computing

stationary (S), moderately decreasing (MD), and rapidly decreasing (RD). This information is passed to the cloud leader whenever there is the need to migrate the VM running the application.

SAM$_k$ interacts with the cluster leader and with the application managers of servers accepting the migration of an application currently running on server $\mathcal{S}_k$. A report sent to the cluster leader includes the list of applications currently running on $\mathcal{S}_k$, their additional demands of over the last reallocation cycle and over a window of w reallocation intervals, and their classification as RI/MI/S/MD/RD over the same window. The scaling decisions are listed in the order of their energy consumption, overhead, and complexity:

(1) Local decision - whenever possible, carry out vertical application scaling using local resources only.

(2) In-cluster, horizontal or vertical scaling - migrate some of the VMs to the other servers identified by the leader; wake-up some of the servers in a sleep state or switch them to one of the sleep states depending on the cluster workload.

(3) Inter-cluster scaling - when the leader determines that the cluster operates at 80% of its capacity with all servers running, the admission control mechanism stops accepting new applications. When the existing applications scale up above 90% of the capacity with all servers running then the cluster leader interacts with the leaders of other clusters to satisfy the requests. This case is not addressed in this paper.

All decisions take into account the current demand for processor capacity, as well as the anticipated load.

I. Local, vertical scaling. The anticipated load at the end of the current and the following scheduling cycle allow the server to continue operating in the optimal regime

$$g_k(t+\tau_k) \leq \gamma_k a_k^{opt,h} \ \& \ \gamma_k a_k^{opt,l} \leq g_k(t+2\tau_k) \leq \gamma_k a_k^{opt,h}. \quad (9)$$

II. In-cluster scaling. The anticipated load could force the server to transition to the suboptimal-low, suboptimal-high, or undesirable-high regimes, respectively:

$$\begin{gathered} g_k(t+\tau_k) \leq \gamma_k \alpha_k^{opt,h} \ \& \ \gamma_k \alpha_k^{sopt,l} \leq g_k(t+2\tau_k) \leq \gamma_k \alpha_k^{opt,l} \\ g_k(t+\tau_k) \leq \gamma_k \alpha_k^{opt,h} \ \& \ \gamma_k \alpha_k^{opt,h} \leq g_k(t+2\tau_k) \leq \gamma_k \alpha_k^{sopt,h} \\ g_k(t+\tau_k) \leq \gamma_k \alpha_k^{opt,h} \ \& \ g_k(t+2\tau_k) > \gamma_k \alpha_k^{sopt,h}. \end{gathered} \quad (10)$$

The leader includes the server in the `WatchList` when Equations 10 are satisfied.

III. Inter-cluster scaling. The anticipated load could force the server to transition to the undesirable-low regime.

$$g_k(t+\tau_k) \leq \gamma_k \alpha_k^{opt,h} \ \ \& \ \ g_k(t+2\tau_k) \leq \gamma_k \alpha_k^{sopt,l}. \quad (11)$$

The leader includes the server in the `MigrationList` in case of Equation 11.

In addition to the lazy approach discussed in (II) when a server operates within the boundaries of the suboptimal-high regime until its capacity is exceeded, one could use an *anticipatory* strategy. To prevent potential SLA violations in the immediate future, in this case we force the migration from the suboptimal-high region as soon as feasible.

A. *Synchronous operation.* The algorithm executed by SAM$-k$ every τ_k units of time is:

```
1. Order applications based on the demand.
   Compute the actual rate of increase or
   decrease in demand over a window of w
   reallocation cycles according to Equation 6.
   Compute servers available capacity.
2. If Equations 9 or 11 are  satisfied send an
    imperative request for application migration.
3. If first or third equations in 10 are satisfied
    send a warning  including application history.
4. Else, reallocate CPU cycles to allow each
   application its largest  rate of increase.
   If time elapsed from the last status report
   is larger than  in-cluster reporting period
   send an update to the leader.
```

A server operating in the undesirable-high regime has a high energy efficiency, but gives us a warning that we should be prepared to activate servers in the sleep states and that VM migration is very likely.

B. *Asynchronous operation.* When a server receives a message related to VM migration, its SAM reacts immediately:

B.1. When $\mathcal{S}_k$ receives a request from the cluster leader to accept the migration or vertical scaling on an application, it first checks that by accepting the request it will still be operating on an optimal regime. If so, it sends an accept message to the leader and to $\mathcal{S}_v$, the server requesting the migration or vertical scaling of application. In the former case it starts one or more VMs for the application; in the later case it waits to receive from $\mathcal{S}_v$ the snapshot of the VM image and then starts the new VMs.

B.2. When, in response to a report of operation in a suboptimal regime, server $\mathcal{S}_k$ receives an accept message for vertical scaling of application $\mathcal{A}_{i,k}$ from another server, $\mathcal{S}_v$, it stops the application, constructs the image of the VM running the application, and then sends it to $\mathcal{S}_v$. For horizontal scaling, it sends $\mathcal{S}_v$ the location of the image.

Cluster management. The leader of a cluster maintains several control structures:

`OptimalList` - includes servers operating in an optimal regime or those in suboptimal regime projected to migrate back to the optimal regime; the list is ordered in the increasing order of computing power. Within a group of servers with similar γ_k, the servers are ordered in the increasing order of available capacity.

`WatchList` - includes servers running in two suboptimal regime and the undesirable-high regimes whose applications are candidates for VM migration.

`MigrationList` - includes servers operating in undesirable regimes. The leader selects candidates for migration and identifies possible targets for migration.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2396059, IEEE Transactions on Cloud Computing

`SleepList` - includes servers in one of the sleep states; the list is ordered on the type of sleep state and then in the increasing order of computing power reflected by the constant γ_k.

The leader performs several functions: admission control for new applications, server consolidation and reactivation, and VM migration.

1. Admission control. When the cluster leader receives a request to accept a new application it computes the available capacity and admits it if the system is not overloaded

$$\frac{d_{\mathcal{C}}(t)}{\sum_{k=1}^{n_{\mathcal{C}}^a}\gamma_k} \leq 0.8 \quad \text{with} \quad d_{\mathcal{C}}(t) = \sum_{k=1}^{n_{\mathcal{C}}^a} d_k(t) \tag{12}$$

with $d_{\mathcal{C}}(t)$ is the available capacity and $n_{\mathcal{C}}^a$ represents the number of servers operating in the optimal and suboptimal regimes, those included in the `OptimalList` and the `WatchList`. If the `SleepList` is not empty, the leader has the choice to place the application on a standby queue and then wake up one or more servers in this list and assign the application to these servers.

2. Server consolidation and re-activation. The leader examines each server in the `MigrationList` and attempts to pair those operating in the lower undesirable regime with the ones in the upper undesirable regime and to migrate applications from the later to the former ones taking into account the application profile. Servers left to operate in the lower undesirable region are then ordered to switch to one of the sleep states, depending on the current system load and the predicted future load. These servers are added to the `SleepList`.

When the server load reaches the *high water mark*, then the leader initiates gradual server transitions from deeper sleep states to the lighter sleep states and reactivation of some servers in state C1.

3. VM migration. A VM can only be migrated at the time when checkpointing is possible and a consistent cut [23] can be defined. The leader acts as a broker for the selection of a target for horizontal or vertical scaling of application $\mathcal{A}$. Once it receives a request for in-cluster scaling it first identifies a potential target; then the two servers, the one sending the request and the one accepting to be the target for horizontal or vertical scaling, negotiate the VM migration. Once an agreement has been reached, the two servers carry out the operation without the intervention of the leader. The target selection is guided by two objectives:

(i) Ensure that the selected target server will be able to accommodate application scaling for an extended period of time, while operating in its optimal regime; this will help reduce the migration costs and the power consumption.

(ii) Keep the user costs low by selecting the least costly server, the one with the lowest γ_k that satisfies condition (i).

The strategy is to consider a window of future intervals, ϕ, and determine the largest possible increase in resource demand of application $\mathcal{A}_i$.

$$a_i(t + \phi \times \lambda_i) = a_i(t) + \phi\lambda_i \tag{13}$$

Then search the *WatchList* for a server operating in the lower suboptimal regime with suitable available capacity

$$d_v(t) > a_i(t + \phi \times \lambda_i). \tag{14}$$

If such a server does not exist, then server $\mathcal{S}_u$ from *SleepList* with the lowest γ_u is selected if it satisfies the conditions

$$a_i(t) \geq \gamma_u \alpha_u^{sopt,l} \quad \text{and} \quad a_i(t + \phi \times \lambda_i) \leq \alpha_u^{sopt,l}. \tag{15}$$

The algorithms assume that the thresholds for normalized performance and power consumption of server $\mathcal{S}_k$ are constant. When the processor supports dynamic voltage and frequency scaling [32] the thresholds, $\alpha_k^{opt,l}(t)$, $\alpha_k^{opt,high}(t)$, $\alpha_k^{sopt,l}(t)$, $\alpha_k^{sopt,high}(t)$, $\beta_k^{opt,l}(t)$, $\beta_k^{opt,high}(t)$, $\beta_k^{sopt,l}(t)$, $\beta_k^{sopt,high}(t)$ will vary in time. The basic philosophy will be the same, we shall attempt to keep every server in an optimal operating regime. An additional complication of the algorithms is that we have to determine if it is beneficial to increase/decrease the power used thus, push up/down the thresholds of the operating regimes of the server. We still want to make most scaling decisions locally.

6 Simulation Experiments

The effectiveness of an energy-aware load balancing and scaling algorithm is characterized by its computational efficiency, the number of operations per Watt of power, and by its negative impact reflected by the potential SLA violations. In our study we shall use the data in Table 2 based on the measurements reported in Table 1 from [4]. These measurements are for a transection processing benchmark, *SPECpower_ssj2008*, thus, the computational efficiency will be the number of transactions per Watt. We define the ideal computational efficiency as the number of transactions when all servers operate at the upper limit of the optimal region, at 80% load, and no SLA violations occur. In this case

$$\left(\frac{T}{P}\right)_{ideal} = \frac{T_{80\%}}{P_{80\%}} = \frac{1049}{235} = 4.46 \; transactions/Watt. \tag{16}$$

We conduct a simulation study to evaluate the effectiveness of the algorithms discussed in Section 5 . The simulation experiments reported in this paper were conducted on the Amazon cloud; an *c3.8xlarge* EC2 instance with 60 G memory and 32 cores was used.

The study will give us some indications about the operation of the algorithm in clusters of different sizes and subject to a range of workloads. The metrics for assessing the effectiveness and the overhead of the algorithms are:

(i) The evolution of the number of servers in each of the five operating regimes as a result of the load migration mandated by the algorithm.

(ii) The computational efficiency before and after the application of the algorithm.

(iii) The ratio of local versus in-cluster scaling decisions during simulation. This reflects the overhead of the algorithm.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2396059, IEEE Transactions on Cloud Computing

Table 3: The effects of application scaling and load balancing algorithm on a system with the parameters described in Table 2. We experimented with two average system loads, 30% and 70% of the server capacity and three different cluster sizes, $10^2, 10^3$, and 10^4. The data before/after the application of the algorithm are: (a) the number of servers in each one of the five operating regimes, $\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3, \mathcal{R}_4, \mathcal{R}_5$ and in the sleep state (slp) (columns 3-8); (b) $\mathbb{P}$ - the average power consumption per processor in Watts (column 9); (c) $\mathbb{T}$ - the performance measured as the average number of transactions per processor (column 10); and (d) the average ratio $\mathbb{T}/\mathbb{P}$ (column 11).

Load	Size	# in $\mathcal{R}_1$	# in $\mathcal{R}_2$	# in $\mathcal{R}_3$	# in $\mathcal{R}_4$	# in $\mathcal{R}_5$	# in slp	$\mathbb{P}$	$\mathbb{T}$	$\mathbb{T}/\mathbb{P}$
30%	10^2	65/0	35/27	0/64	0/2	0/3	0/4	188/197	264/746	1.4/3.8
	10^3	550/0	450/300	0/598	0/20	0/64	0/18	190/214	442/825	2.3/3.8
	10^4	5500/0	4500/3050	0/5950	0/50	0/319	0/631	190/203	442/771	2.3/3.8
70%	10^2	0/0	0/58	20/35	80/2	0/5	0/0	234/204	1,052/742	4.5/3.6
	10^3	0/0	0/430	190/490	810/20	0/56	0/4	234/215	1,054/823	4.5 /3.7
	10^4	0/0	0/4000	2000/4500	8000/250	0/233	0/17	235/193	1,052/715	4.5/3.7

For simplicity we chose only two sleep states C_3 and C_6 in the simulation. If the load of the cluster is more than 60% of the cluster capacity we do not choose the C_6 state because the probability that the system will require additional resources in the next future is high. Switching from the C_6 state to C_0 requires more energy and takes more time. On the other hand, when the cluster load is less than 60% of its capacity we choose the C_6 state because it is unlikely that the server will be reactivated in the next future.

The simulation uses Amazon Web Services (AWS). AWS offers several classes of services; the servers in each class are characterized by the architecture, CPU execution rate, main memory, disk space, and I/O bandwidth. The more powerful the server, the higher the cost per hour for the class of service.

Table 3 summarizes the effects of application scaling and load balancing algorithm on a cluster when the parameters of the servers are given in Table 2 and the application is a transaction processing system. In a transaction processing system there is no workload migration, a front-end distributes the transactions to the servers thus, we did not include migration costs. To asses the power consumption and the performance measured by the number of transactions we use average values for each regime. Recall that in this case the boundaries of the five operating regimes are given in Equation 6.

The effect of the system load. In [27] we report on simulation experiments designed to evaluate the effectiveness of the algorithms for load balancing and energy optimization during application scaling. One of the questions we addressed was whether the system load has an effect on the resource management strategy to force the servers in a cluster to operate within the boundaries of the optimal regime. The experiments we report in this paper are for clusters with $10^2, 10^3$, and 10^4 servers consisting of multiple racks of a WSC. For each cluster size we considered two load distributions:

(i) Low average load - an initial load uniformly distributed in the interval $20-40\%$ of the server capacity. Figure 2 (a) shows the distribution of the number of servers in the five operating regimes for clusters with 10^4 servers, before and after load balancing; the distributions for 10^2 and 10^3 servers in a cluster are similar. When the average server load is 30% of their capacity, the algorithm is very effective; it substantially improves the energy efficiency to 3.8 from as low as 1.4. After load balancing, the number of servers in the optimal regime increases from 0 to about 60% and a fair number of servers are switched to the sleep state.

(ii) High average load - initial server load uniformly distributed in the $60-80\%$ of the server capacity. Figure 2 (b) shows that when the average server load is 70% of their capacity, the average computational efficiency decreases from about 4.5 to 3.6 as many servers, about 80% of them, are forced from $\mathcal{R}_4$ to the $\mathcal{R}_2$ and $\mathcal{R}_3$ regimes to reduce the possibility of SLA violations. Initially, no servers operated in the $\mathcal{R}_5$ regime. In this experiment we did not use the anticipatory strategy and allowed servers to operate in the $\mathcal{R}_5$ regime after load balancing; as a result, a small fraction of servers ended up in the $\mathcal{R}_5$ regime. There is a balance between computational efficiency and SLA violations; the algorithm can be tuned to maximize computational efficiency or to minimize SLA violations according to the type of workload and the system management policies.

These results are consistent with the ones reported in [27] for smaller cluster sizes, 20, 40, 60, and 80 servers. This shows that the algorithms operate effectively for a wide range of cluster sizes and for lightly, as well as, heavily loaded systems.

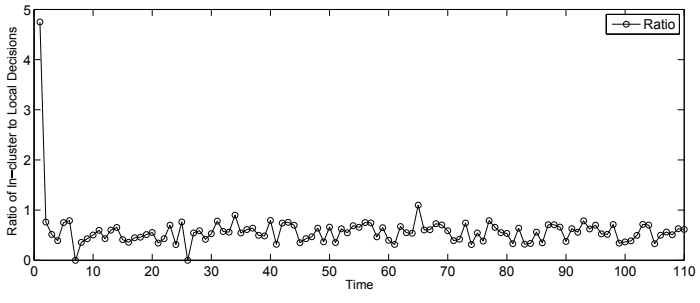
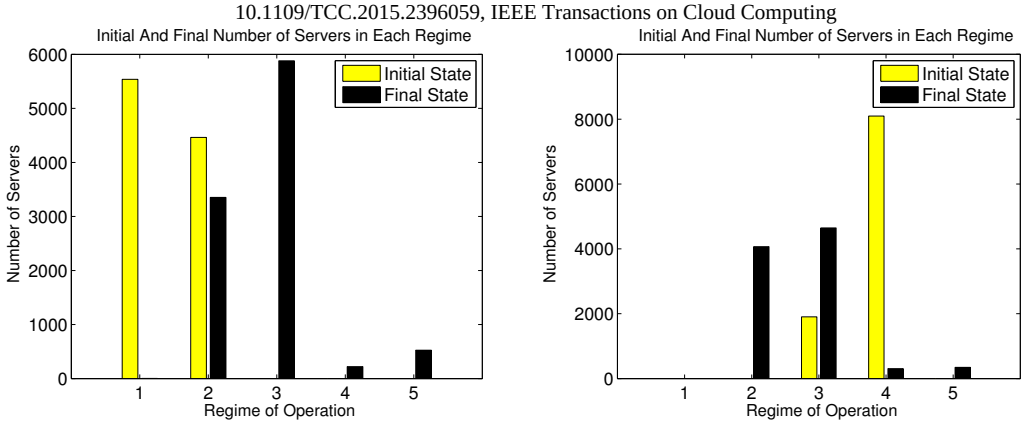


Figure 4: The ratio of in-cluster to local decisions in response to scaling requests versus time for a cluster with 40 servers when the average workload is 50% of the server capacity.

High-cost versus low-cost application scaling. Cloud

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2396059, IEEE Transactions on Cloud Computing



(a) Cluster size: 10^4. Average load: 30%

(b) Cluster size 10^4. Average load: 70%

Figure 2: The effect of average server load on the distribution of the servers in the five operating regimes, $\mathcal{R}_1$, $\mathcal{R}_2$, $\mathcal{R}_3$, $\mathcal{R}_4$ and $\mathcal{R}_5$, before and after energy optimization and load balancing. Average load: (a) 30% and (b) 70%. Cluster size: 10^4. Similar results are obtained for cluster size 10^2 and 10^3.

Table 4: Average and standard deviation of in-cluster to local decisions ratio for 30% and 70% average server load for three cluster sizes, 10^2, 10^3, and 10^4.

Cluster size	Average load	Average ratio	Standard deviation	Average load	Average ratio	Standard deviation
10^2	30%	0.69	0.57	70%	0.51	0.89
10^3	30%	0.33	0.21	70%	0.58	0.92
10^4	30%	0.49	0.27	70%	0.53	0.98

elasticity allows an application to seamlessly scale up and down. In the next set of simulation experiments we investigate horizontal and vertical application scaling. *Horizontal scaling* requires the creation of additional VMs to run the application on lightly loaded servers. In-cluster, horizontal scaling incurs higher costs than vertical scaling for load balancing. The higher costs in terms of energy consumption and time are due to the communication with the leader to identify the potential targets and then to transport the VM image to one or more of them. Local, *vertical scaling*, allows the VM running an application to acquire additional resources from the local server; local vertical scaling has lower costs, but it is only feasible if the server has sufficient free capacity.

The average ratio of high-cost in-cluster horizontal scaling to low-cost local vertical scaling for a transient period of 40 reallocation intervals are summarized in Table 4 and in Figures 3 (a), (b), (c), (d), and (e). When the average workload is 30% the algorithm works better; the larger the cluster, the more effectively the algorithm handles the application scaling as most decisions are done locally. After 40 reallocation intervals the ratios for the two average workload are 0.3 and 0.7, respectively, for cluster size 10^4; this shows that at higher workload VM migrations are more intense. The trends are different for the two cases, the ratio decreases in time when the average load is 30% and increases when the average load is 70%; we expect these trends to continue in the steady-state regime.

We carried out measurements of computational efficiency on a system with a 3GHz Intel Core i7 with 8 cores and a solid state disk. The system was running under OS X Yosemite. The application used for our measurements runs under `Xcode 6`, an integrated development environment containing a suite of software tools for OS X and iOS. The application is I/O intensive, it reads a record from the secondary storage carries out some trivial computations and writes back the record. `Xcode` allowed us to measure the normalized performance and the corresponding normalized power consumption and thus, to calculate the computational efficiency at the boundaries of the five operating regimes described by Equation 6. `Xcode` reports only the power used by the cores running the application, so the average computational efficiency we were able to measure refers only to the processor, rather than the entire system.

We then considered a cloud infrastructure consisting of $10,000$ servers identical with the system we have measured. We used the distribution of the servers in each of the five regimes in Table 3 to compute the computational efficiency before and after the application of the algorithm.

The results are summarized in Table 5. We see that the average computational efficiency decreases from 1.42 in $\mathcal{R}_2$ to 1.245 in $\mathcal{R}_3$, and 1.055 in $\mathcal{R}_4$. As a result, the computational efficiency due to application of our algorithm increases only from 1.03 to 1.21 for the lightly loaded system and shows a modest increase from 1.09 to 1.18 for the heavy load case.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2396059, IEEE Transactions on Cloud Computing

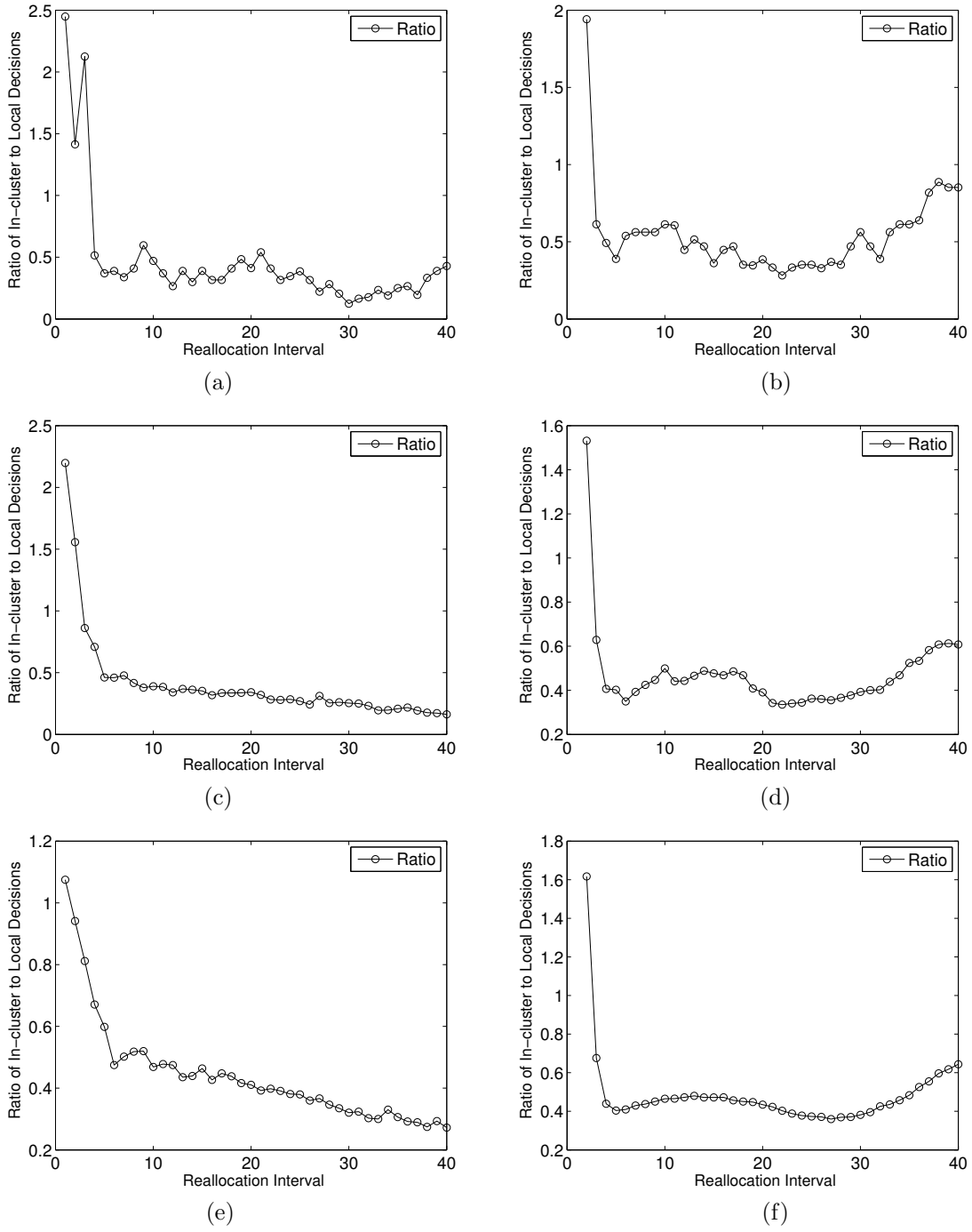


Figure 3: Time series of in-cluster to local decisions ratios for a transient period of 40 reallocation intervals. Average load: 30% of the server capacity in (a), (c), and (e); 70% in (b), (d), and (f). The cluster size: 10^2 in (a) and (b); 10^3 in (c) and (d); 10^4 in (e) and (f). After 40 reallocation intervals almost double rations when the load is 70% versus 30%.

As noted in Section 2, the processors consume less than one-third of their peak power at the very-low load thus, the actual improvement due to the application of our algorithm should be considerably higher.

Finally, we attempted to carry out measurements in a realistic cloud environment and we investigated the possibility of using EC2 instances. First, we realized that no Mac OS-based AMIs (Amazon Machine Images) are supported, so we could not use `Xcode`. We then discovered that the AWS virtual machine monitors prevent both `Linux` and `Ubuntu` AMIs to collect information related to power consumption. As a result, tools such as `dmidecode` or `lshw` used to report hardware information by monitoring kernel data structures return the fields related to power consumption as *unknown*. This lack of transparency makes the investigation of energy consumption in cloud environments a rather challenging task.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2396059, IEEE Transactions on Cloud Computing

Table 5: The effects of application scaling and load balancing algorithm on a system with 10^4 servers 3GHz Intel Core i7. Shown are the number of servers in each one of the five operating regimes before and after the application of the algorithm according to Table 3 and the average computational efficiency in each regime $\bar{C}_{ef}^{\mathcal{R}_i}$, $1 \leq i \leq 5$ determined by our measurements. $\bar{C}_{ef}$ shows the average computational efficiency before and after the application of the algorithm.

Load	# in $\mathcal{R}_1$ $\bar{C}_{ef}^{\mathcal{R}_1} = 0.725$	# in $\mathcal{R}_2$ $\bar{C}_{ef}^{\mathcal{R}_2} = 1.420$	# in $\mathcal{R}_3$ $\bar{C}_{ef}^{\mathcal{R}_3} = 1.245$	# in $\mathcal{R}_4$ $\bar{C}_{ef}^{\mathcal{R}_4} = 1.055$	# in $\mathcal{R}_5$ $\bar{C}_{ef}^{\mathcal{R}_5} = 1.050$	$\bar{C}_{ef}$
30%	5500/0	4500/3050	0/5950	0/50	0/319	1.03/1.21
70%	0/0	0/4000	2000/4500	8000/250	0/233	1.09/1.18

7 Conclusions & Future Work

The realization that power consumption of cloud computing centers is significant and is expected to increase substantially in the future motivates the interest of the research community in energy-aware resource management and application placement policies and the mechanisms to enforce these policies. Low average server utilization [30] and its impact on the environment [8] make it imperative to devise new energy-aware policies which identify optimal regimes for the cloud servers and, at the same time, prevent SLA violations.

A quantitative evaluation of an optimization algorithm or an architectural enhancement is a rather intricate and time-consuming process; several benchmarks and system configurations are used to gather the data necessary to guide future developments. For example, to evaluate the effects of architectural enhancements supporting Instruction-level or Data-level Parallelism on the processor performance and their power consumption several benchmarks are used [17]. The results show different numerical outcomes for the individual applications in each benchmark. Similarly, the effects of an energy-aware algorithm depend on the system configuration and on the application and cannot be expressed by a single numerical value.

Research on energy-aware resource management in large-scale systems often use simulation for a quasi-quantitative and, more often, a qualitative evaluation of optimization algorithms or procedures. As stated in [4] "First, they (WSCs) are a new class of large-scale machines driven by a new and rapidly evolving set of workloads. Their size alone makes them difficult to experiment with, or to simulate efficiently." It is rather difficult to experiment with the systems discussed in this paper and this is precisely the reason why we choose simulation.

The results of the measurements reported in the literature are difficult to relate to one another. For example, the wake-up time of servers in the sleep state and the number of servers in the sleep state are reported for the AutoScale system [10]; yet these figures would be different for another processor, system configuration, and application.

We choose computational efficiency, the ratio of the amount of normalized performance to normalized power consumption, as the performance measure of our algorithms. The amount of useful work in a transition processing benchmark can be measured by the number of transactions, but it is more difficult to assess for other types of applications. SLA violations in a transaction processing benchmark occur only when the workload exceeds the capacity of all servers used by the application, rather than the capacity of individual servers. Thus, in our experiment there are no SLA violation because there are servers operating in low-load regimes.

We need to balance computational efficiency and SLA violations; from Table 3 we see that the computational efficiency increases up to $3.6 - 3.8$, while the optimum is 4.46 transactions/Watt. Figure 2 (b) and Table 3 show that the computational efficiency decreases after the application of the algorithm to a system with the average load 70% because servers operating in the suboptimal-high regime are forced to reduce their workload. The *lazy* approach discussed in Section 5 would eliminate this effect.

Even the definition of an ideal case when a clairvoyant resource manager makes optimal decisions based not only on the past history, but also on the knowledge of the future can be controversial. For example, we choose as the ideal case the one when all servers operate at the upper boundary of the optimal regime; other choices for the ideal case and for the bounds of the five regimes could be considered in case of fast varying, or unpredictable workloads.

The five-regime model introduced in this paper reflects the need for a balanced strategy allowing a server to operate in an optimal or near-optimal regime for the longest period of time feasible. A server operating in the optimal regime is unlikely to request a VM migration in the immediate future and to cause an SLA violation, one in a sub-optimal regime is more likely to request a VM migration, while one in the undesirable-high regime is very likely to require VM migration. Servers in the undesirable-low regime should be switched to a sleep state as soon as feasible.

The model is designed for clusters built with the same type of processors and similar configurations; the few parameters of the model are then the same for all the servers in the cluster. The clustered organization allows an effective management of servers in the sleep state as they should be switched proactively to a running state to avoid SLA violations. It also supports effective admission control, capacity allocation, and load balancing mechanisms as the cluster leader has relatively accurate information about the available capacity of individual servers in the cluster.

Typically, we see a transient period when most scaling decisions require VM migration, but in a steady-state, local decisions become dominant. Table 4 shows that the average

system load affects the average ratio of in-cluster to local decisions thus, the migration costs, increases from 0.33 to 0.55 for a cluster with 10^3 servers when the average system load increases from 30% to 70%. As pointed out in [4] there is a real need for energy benchmarks such as the one in [29] which could be used to guide the design choices for new systems.

From the large number of questions posed by energy-aware load balancing policies we restrict our analysis to the conditions when a server should be switched to a sleep state and the choice of the sleep state, and how to chose the target server where to move a VM. The mechanisms for load balancing and application scaling policies are consistent with the requirements discussed in Section 3, scalability, effectiveness, practicality, and consistency with global system objectives.

Though designed specifically for the IaaS cloud delivery model, the algorithms discussed in Section 5 can be adapted for the other cloud delivery models. Typically, PaaS applications run for extended periods of time and the smallest set of servers operating at an optimal power level to guarantee the required turnaround time can be determined accurately.

Our future work will evaluate the overhead and the limitations of the algorithm proposed in this paper; it will also include the implementation of a Server Application Manager and the evaluation of the overhead for the algorithm proposed in this paper. The algorithm will be incorporated in the self-management policies introduced in [24].

8 Acknowledgments

The authors are grateful for the constructive comments of anonymous reviewers.

References

[1] D. Ardagna, B. Panicucci, M. Trubian, and L. Zhang. "Energy-aware autonomic resource allocation in multi-tier virtualized environments." *IEEE Trans. on Services Computing,* **5**(1):2–19, 2012.

[2] J. Baliga, R.W.A. Ayre, K. Hinton, and R.S. Tucker. "Green cloud computing: balancing energy in processing, storage, and transport." *Proc. IEEE*, **99**(1):149-167, 2011.

[3] L. A. Barroso and U. Hözle. "The case for energy-proportional computing." *IEEE Computer,* **40**(12):33–37, 2007.

[4] L. A. Barossso, J. Clidaras, and U.Hözle. *The Datacenter as a Computer; an Introduction to the Design of Warehouse-Scale Machines.* (Second Edition). Morgan & Claypool, 2013.

[5] A. Beloglazov, R. Buyya "Energy efficient resource management in virtualized cloud data centers." *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Comp.*, 2010.

[6] A. Beloglazov, J. Abawajy, R. Buyya. "Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing." *Future Generation Computer Systems*, **28**(5):755-768, 2012.

[7] A. Beloglazov and R. Buyya. "Managing overloaded hosts for dynamic consolidation on virtual machines in cloud centers under quality of service constraints." *IEEE Trans. on Parallel and Distributed Systems*, **24**(7):1366-1379, 2013.

[8] M. Blackburn and A. Hawkins. "Unused server survey results analysis." *www.thegreengrid.org/media/White Papers/Unused%20Server%20Study_WP_101910_v1.ashx?lang=en* (Accessed on December 6, 2013).

[9] M. Elhawary and Z. J. Haas. "Energy-efficient protocol for cooperative networks." *IEEE/ACM Trans. on Networking,* **19**(2):561–574, 2011.

[10] A. Gandhi, M. Harchol-Balter, R. Raghunathan, and M.Kozuch. "AutoScale: dynamic, robust capacity management for multi-tier data centers." *ACM Trans. on Computer Systems*, **30**(4):1–26, 2012.

[11] A. Gandhi, M. Harchol-Balter, R. Raghunathan, and M.Kozuch. "Are sleep states effective in data centers?" *Proc. Int. Conf. on Green Comp.*, pp. 1–10, 2012.

[12] D. Gmach, J. Rolia, L. Cherkasova, G. Belrose, T. Tucricchi, and A. Kemper. "An integrated approach to resource pool management: policies, efficiency, and quality metrics." *Proc. Int. Conf. on Dependable Systems and Networks,* pp. 326–335, 2008.

[13] Google. "Google's green computing: efficiency at scale." *http://static.googleusercontent.com/external_content/untrusted_dlcp/www.google.com/en/us/green/pdfs/google-green-computing.pdf* (Accessed on August 29, 2013).

[14] V. Gupta and M. Harchol-Balter. "Self-adaptive admission control policies for resource-sharing systems." *Proc. 11th Int. Joint Conf. Measurement and Modeling Computer Systems (SIGMETRICS'09)*, pp. 311–322, 2009.

[15] K. Hasebe, T. Niwa, A. Sugiki, and K. Kato. "Power-saving in large-scale storage systems with data migration." *Proc IEEE 2nd Int. Conf. on Cloud Comp. Technology and Science,* pp. 266–273, 2010.

[16] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown. "ElasticTree: saving energy in data center networks." *Proc. 7th USENIX Conf. on Networked Systems Design and Implementation*, pp. 17–17, 2011.

[17] J. L. Hennessy and D. A. Patterson. *Computer Architecture; A Quantitative Approach, 5th Edition.* Morgan Kaufmann, 2012.

[18] Hewlet-Packard/Intel/Microsoft/Phoenix/Toshiba. "Advanced configuration and power interface specifications, revision 5.0" *http://www.acpi.info/DOWNLOADS/ACPIspec50.pdf*, 2011. (Accessed on November 10, 2013).

[19] J. G. Koomey. "Estimating total power consumption by servers in the US and world." *http://hightech.lbl.gov/documents/data_centerssvrpwrusecompletefinal.pdf* (Accessed on May 11, 2013).

[20] J.G. Koomey, S. Berard, M. Sanchez, and H. Wong. "Implications of historical trends in the energy efficiency of computing." *IEEE Annals of Comp.*, **33**(3):46–54, 2011.

[21] E. Le Sueur and G. Heiser. "Dynamic voltage and frequency scaling: the laws of diminishing returns." *Proc. Workshop on Power Aware Computing and Systems, HotPower'10*, pp. 2–5, 2010.

[22] B. Li; J. Li; J. Huai; T. Wo; Q. Li; L. Zhong. "EnaCloud: an energy-saving application live placement approach for cloud computing environments." *Cloud Comp., 2009. CLOUD '09, pp.17 - 24, 2009.*

[23] D. C. Marinescu. *Cloud Computing; Theory and Practice.* Morgan Kaufmann, 2013.

[24] D. C. Marinescu, A Paya, and J.P. Morrison. "Coalition formation and combinatorial auctions; applications to self-organization and self-management in utility comput ing." http: //arxiv.org/pdf/1406.7487v1.pdf, 2014.

[25] C. Mastroianni, M. Meo, G. Papuzzo. " Probabilistic consolidation of virtual machines in self-organizing cloud data centers." *IEEE Trans. on Cloud Computing,* **1**(2):215–228, 2013.

[26] NRDC and WSP 2012. "The carbon emissions of server computing for small-to medium-sized organization - a performance study of on-premise vs. the cloud." *http://www.wspenvironmental.com/media/docs/ourlocations/usa/NRDC-WSP_Cloud_Computing.pdf* October 2012 (Accessed on November 10, 2013).

[27] A. Paya and D. C. Marinescu. "Energy-aware load balancing policies for the cloud ecosystem." http: //arxiv.org/pdf/1307.3306v1.pdf, December 2013.

[28] C. Preist and P. Shabajee. "Energy use in the media cloud." *Proc IEEE 2nd Int. Conf. on Cloud Comp. Technology and Science,* pp. 581–586, 2010.

[29] SPEC "SPECpower_ssj2008 benchmark." *https://www.spec.org/power_ssj2008/* (Accessed on May 15, 2014).

[30] B. Snyder. "Server virtualization has stalled, despite the hype." *http://www.infoworld.com/print/146901* (Accessed on December 6, 2013).

[31] B. Urgaonkar and C. Chandra. "Dynamic provisioning of multi-tier Internet applications." *Proc. 2nd Int. Conf, on Automatic Comp.*, pp. 217–228, 2005.

[32] H. N. Van, F. D. Tran, and J.-M. Menaud. "Performance and power management for cloud infrastructures." *Proc. IEEE 3rd Int. Conf. on Cloud Comp.*, pp. 329–336, 2010.

[33] A. Verma, G. Dasgupta, T. K. Nayak, P. De, and R. Kothari. "Server workload analysis for power minimization using consolidation." *Proc. USENIX'09 Conf.*, pp.28–28, 2009.

[34] S. V. Vrbsky, M. Lei, K. Smith, and J. Byrd. "Data replication and power consumption in data grids." *Proc IEEE 2nd Int. Conf. on Cloud Computing Technology and Science,* pp. 288–295, 2010.

[35] ZDNet. "Infrastructucture is not a differentiator." *http://www.zdnet.com/amazon-cto-werner-vogels-infrastructure-is-not-a-differentiator-7000014213* (Accessed on August 29, 2013).

Ashkan Paya is a Ph.D. candidate in the Electrical Engineering and Computer Science Department at University of Central Florida pursuing his degree in Computer Science. He graduated from Sharif University of Technology in Tehran, Iran, with a BS Degree in the same major in 2011. His research interests are in the area of resource management in large-scale systems and cloud computing.

Dan C. Marinescu. During the period 1984-2001 Dan Marinescu was an Associate and then Full Professor in the Computer Science Department at Purdue University in West Lafayette, Indiana. Since August 2001 he is a Provost Professor of Computer Science at University of Central Florida. He has published more than 220 papers in referred journals and conference proceedings and several books including *Cloud Computing: Theory and Practice*, Morgan Kaufmann, 2013.