Network-aware energy-efficient virtual machine management in distributed Cloud infrastructures with on-site photovoltaic production

Benjamin Camus, Fanny Dufossé, Anne Blavette, Martin Quinson, Anne-Cécile Orgerie

Network-aware energy-efficient virtual machine management in distributed Cloud infrastructures with on-site photovoltaic production

Benjamin Camus*, Fanny Dufossé†, Anne Blavette‡, Martin Quinson* and Anne-Cécile Orgerie*
*Univ. Rennes, Inria, CNRS, IRISA, Rennes, France
Email: {benjamin.camus, martin.quinson, anne-cecile.orgerie}@irisa.fr
†Inria, LIG, Grenoble, France
Email: fanny.dufosse@inria.fr
‡Univ. Rennes, CNRS, SATIE, Rennes, France
Email: anne.blavette@ens-rennes.fr

***Abstract*—Distributed Clouds are nowadays an essential component for providing Internet services to always more numerous connected devices. This growth leads the energy consumption of these distributed infrastructures to be a worrying environmental and economic concern. In order to reduce energy costs and carbon footprint, Cloud providers could resort to producing on-site renewable energy, with solar panels for instance. In this paper, we propose NEMESIS: a Network-aware Energy-efficient Management framework for distributEd cloudS Infrastructures with on-Site photovoltaic production. NEMESIS optimizes VM placement and balances VM migration and green energy consumption in Cloud infrastructure embedding geographically distributed data centers with on-site photovoltaic power supply. We use the Simgrid simulation toolbox to evaluate the energy efficiency of NEMESIS against state-of-the-art approaches.**

***Index Terms*—Distributed cloud computing, renewable energy, energy-efficient scheduling, on-site production, network-aware migration, consolidation.**

I. Introduction

The growing appetite of Internet services for Cloud resources leads to a consequent increase in data center (DC) facilities worldwide. This increase directly impacts the electricity bill of Cloud providers. Indeed, electricity is currently the largest part of the operation cost of a DC [1]. Resource over-provisioning, energy non-proportional behavior of today's servers, and inefficient cooling systems have been identified as major contributors to the high energy consumption in DCs [2].

On-site renewable energy production and geographical energy-aware load balancing of virtual machines (VM) allocation can be associated to lower the brown (i.e. not renewable) energy consumption of DCs. Yet, combining these two approaches remains challenging in current distributed Clouds. Indeed, the variable and/or intermittent behavior of most renewable sources – like solar power for instance – is not correlated with the Cloud energy consumption, that depends on physical infrastructure characteristics and fluctuating unpredictable workloads. Moreover, while literature tackles this issue, existing work often does not consider heterogeneous VMs running over several time slots, network constraints (bandwidth, latency and topology) of distributed Clouds, the energy cost of turning on/off servers and of migrating VMs, and the stochastic behavior of renewable supplies.

The main contribution of this paper is NEMESIS: a Network-aware Energy-efficient Management framework for distributEd cloudS Infrastructures with on-Site photovoltaic production. The originality of NEMESIS lies in its combination of a greedy VM allocation algorithm, a network-aware live-migration algorithm, a dichotomous consolidation algorithm and a stochastic model of the renewable energy supply in order to optimize both green and brown energy consumption of a distributed cloud infrastructure with on-site renewable production. Our solution employs a centralized resource manager to schedule VM migrations in a network-aware and energy-efficient way, and consolidation techniques distributed in each data center to optimize the Cloud's overall energy consumption.

NEMESIS has been implemented on the SimGrid toolkit [3] using real Cloud traces. This simulator embeds accurate energy consumption models and fine-grain VM and live-migration abstractions. The simulation-based evaluation shows that NEMESIS outperforms classical greedy allocations and algorithms from the literature with a realistic workload that comprises heterogeneous VMs and does not exhibit day/night patterns.

The rest of the paper is organized as follows. Section II presents related work. Section III details the employed models and Section IV describes our NEMESIS approach. Simulation results are presented in Section V. Section VI concludes this work and introduces future work.

II. Related work

As their increasing electricity bill raises environmental issues, Cloud providers resort more and more to renewable energy sources [4]. On one hand, the renewable production can be adjusted through the use of energy storage devices. Yet, this solution is costly and far from ideal as these devices present charge and discharge maximal rates, depth of discharge lower

bounds,as well as strong aging effects and non-negligible energy losses ratios. Furthermore, green energy availability highly depends on the data centers' location that is fixed upon construction.

On the other hand, Cloud providers can try to adjust the workload to the energy production. For instance, opportunistic scheduling aims at postponing Cloud's workload during low-production periods to wait for renewable energy availability [5]. Distributed Cloud systems can perform geographic load balancing and follow-the-sun resource management in order to increase the green energy use [6]. It consists in scheduling the virtual machine allocations to data centers where renewable energy is available. Such techniques can also be combined with consolidation algorithms that optimize the number of used resources by migrating virtual machines (VM) and switch off unused resources [7]. Yet, Cloud data centers are required to provide high availability to their customers, and consequently, parts of the workload cannot be reshaped.

While follow-the-sun and consolidation techniques can save energy in distributed Cloud infrastructures, existing frameworks often do not consider network constraints [6]. Indeed, as Cloud traffic demands diversify, network resources, inside and in-between the data centers, are often stretched to their limits and, in many cases, become a performance bottleneck [8]. If not carefully taken into account, network can be a major issue for energy-efficient resource management techniques and it can make them unfeasible in practice.

III. Model

In this section, we describe the considered Cloud model and its underlying assumptions. We consider a distributed Cloud infrastructure comprising several data centers (DC) geographically distributed and connected through an IP telecommunication network. The DCs host a given amount of servers which are homogeneous in terms of memory, number of cores and energy consumption. On the contrary to other works in the literature, we take into account the complete IP network topology, i.e. the different links (with their latency and bandwidth constraints) and their topology.

Each DC produces its own green energy thanks to photovoltaic panels (PV). The energy production of the PV is not known in advance as it strongly depends on the meteorological context of each DC. When the local green energy production of a DC is not sufficient, the traditional, regular electrical grid is in charge of powering the Cloud. Following the worst case scenario, all the supply coming from the regular grid is considered as brown energy.

The user management of the Cloud is assumed to be centralized. At each time slot of 5min, heterogeneous VMs (in terms of memory, CPU and execution time requirements) are submitted to the Cloud. They will run for a given period of time *a priori* known. The Cloud manager is free to locate a VM on any server with enough resources to run the VM (i.e. no over-commitment). The VM location can be changed at runtime by using live migrations.

We consider four states for a server: POWERING_ON, POWERING_OFF, ON and OFF. Before being ON (resp. OFF), a server remains in the POWERING_ON (resp. POWERING_OFF) state for a given fixed duration. When the servers are ON, we assume their power consumption to change linearly with CPU usage [9]. Each of the three other states is associated with a static power consumption.

The power consumption of a server is the sum of a fixed part P_{idle} and a second part proportional to the number n_c of running CPUs $n_c \times P_c$. It consumes electric power P_{ON} when turning on, and P_{OFF} when turning off. The number of servers on, at time slot t, on data center DC_i, is denoted $U_i(t)$. Thus the static power of DC_i cost is $U_i(t) \times P_{idle}$. The migration of a VM during t will consume an **average** power of $P_c \times \frac{t_M}{5min}$ during the full duration of t (i.e. 5min). Note that we assume here that VM migrations are shorter than the duration of a timeslot (the algorithms we propose in the following section guarantee this point). Finally, we count the switch ON/OFF of servers: $U_i^+(t) \times P_{ON} + U_i^-(t) \times P_{OFF}$, where $U_i^+(t) = \max(0, U_i(t) - U_i(t-1))$ and $U_i^-(t) = \max(0, U_i(t-1) - U_i(t))$.

In order to determine the migration time of a VM, we consider the live migration model of [10]. The live migration of a VM from a server A to a server B comprises several steps: reservation, iterative pre-copy, stop-and-copy, commitment and activation. In our migration model, we consider the execution time of steps reservation and activation to be negligible. For sake of simplicity, we also average the duration of dirty-pages iterative transfers as this highly depends on the application running on the VM. Consequently, the migration of a VM consists in (1) transferring all the pages, (2) sending a message to notify the end of the stop-and-copy step, and (3) sending the commitment message. Hence, following the TCP/IP model of [10], the time T_m to migrate a VM is equal to: $T_m = 3L_m + {}^{S_m}/_{\rho_m}$ with S_m the memory size of the VM, L_m the latency of the route used for the transfer (equals to 13.01 times the sum of the physical latency of the links [10]), and ρ_m the throughput of the transfer. The transfer throughput can be limited by the physical bandwidth or the latency of the route. Therefore, it is equal to: $\rho_m = \min\left(0.92B_m, {}^{W}/_{2L_m}\right)$ with B_m the physical bandwidth and $W = 4194304$B, the TCP maximum window size. Please, note that [10] determined the values of the different constants of the model with rigorous experiments made on real TCP network. We consider that the migration requires a full core on the destination server. Thus during the migration, as the source server still runs the VM, both the source and the destination servers consume energy.

IV. Our approach

Our approach consists in a centralized Algorithm 1, NEMESIS, that operates four main steps. In the first step, further detailed in Section IV-B, Algorithm 2 allocates waiting VMs to servers on DCs according to the availability of servers and to the on-site green electricity production. This algorithm mainly focuses on allocating the VMs on the least possible number of servers. Then, Algorithm 3 revisits decisions of Algorithm 2.

Algorithm 1: General algorithm

```
Allocate new VMs;                    (Algorithm 2)
Migrate pre-allocated VMs;           (Algorithm 3)
Migrate running VMs;                 (Algorithm 5)
for 1 ≤ i ≤ M do
    Consolidate DC_i                 (Algorithm 6)
```

Then, as detailed in Section IV-C, Algorithm 5 determines migration of running VMs between data centers. Finally in Section IV-D, on each DC, Algorithm 6 determines if some servers can be turned off after intra DC migration.

A. Expected brown consumption computation

We first explain how to compute and forecast the brown consumption of the Cloud. Using our server model, we can compute the power consumption P of a DC. This electricity is provided by both local PVs, and by brown electricity. Similarly to [11], the PV production is modeled by a normal law $\mathcal{N}(Eg_i(t), p_i(t))$ truncated in 0. This model is based on the common assumption that the forecast error of PV production follows a normal distribution [12]. Then, for any time slot, the expected brown consumption is:

$$Ec_i(P,t) = (P - Eg_i(t))\frac{\Phi(P) - \Phi(0)}{1 - \Phi(0)} - p_i(t)^2 \frac{\phi(0) - \phi(P)}{1 - \Phi(0)},$$

with $\phi(x) = \frac{1}{p_i(t)\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x - Eg_i(t)}{p_i(t)}\right)^2}$, and $\Phi(x) = \frac{1}{2}\left(1 + \text{erf}\left(\frac{x - Eg_i(t)}{p_i(t)\sqrt{2}}\right)\right)$. Details of computations are given in [11]. Concerning the expected remaining green energy, we obtain the following result. The expected green power used is:

$$\begin{aligned} EgU_i(t) &= P - Ec_i(P,t) \\ &= P - (P - Eg_i(t))\tfrac{\Phi(P) - \Phi(0)}{1 - \Phi(0)} + p_i(t)^2 \tfrac{\phi(0) - \phi(P)}{1 - \Phi(0)} \end{aligned}$$

and the expected green power production is $EgP_i(t) = Eg_i(t) + p_i(t)^2 \phi(0)/1 - \Phi(0)$

Then, the expected remaining green power is:

$$\begin{aligned} EgR_i(t) &= EgP_i(t) - EgU_i(t) \\ &= Eg_i(t) - P + (P - Eg_i(t))\tfrac{\Phi(P) - \Phi(0)}{1 - \Phi(0)} \\ &\quad + p_i(t)^2 \tfrac{\phi(0)}{1 - \Phi(0)} - p_i(t)^2 \tfrac{\phi(0) - \phi(P)}{1 - \Phi(0)} \\ &= (Eg_i(t) - P)\tfrac{1 - \Phi(P)}{1 - \Phi(0)} + p_i(t)^2 \tfrac{\phi(P)}{1 - \Phi(0)} \end{aligned}$$

These formulas permit to compute the expected brown energy consumption and the remaining green power for the next time slot only. However, when allocating VMs, the brown energy consumption needs to be evaluated over the whole execution time. Nevertheless, evaluation on all time slots could be too expensive to compute for very long VMs. We thus define a parameter n_{eval}. This parameter corresponds to the number of time slots where our algorithm evaluates the expected power consumption for a given computation. For example for a VM v of execution time t_e, we will evaluate the brown power consumed at current time slot t and then at $n_{eval} - 1$ other time slots to evaluate the global brown extra cost related to this VM, for a given DC. We limit this evaluation to the current day light. This evaluation does not make sense during night as PV are not producing energy, and our forecast model does not predict the next day weather. We define $t_{max} = min(t_{sunset}, t + t_e)$ as the last time slot for our evaluation, and $t_k = t + k \times \frac{t_{max} - t}{n_{eval}}$, for k between 0 and $n_{eval} - 1$. Evaluations over less than n_{eval} time slots are done over all time slots, for example if $t_e \leq n_{eval}$.

Thus, if an evaluation over many time slots is requested at time slot t, then the algorithm will compute the brown power consumption for many time slots. The evaluation for the next time slot has been detailed above. For further time slots, the power consumption of the DC will depend on tasks allocation and migrations done in between. However, the computation of the requested power is done based only on currently allocated (or pre-allocated) VMs, as if no more VMs would be submitted any more. Considering VMs finishing at previous time slot $t_k - 1$ and still running at time slot t_k, the power consumption is computed, and the expected power consumption is determined.

B. Allocation

In the following, we say that a server is *incomplete* if it is not yet fully loaded (without considering overcommit). Algorithm 2 pre-allocates all the incoming new VMs at current time slot. These pre-allocations are then revisited by Algorithm 3.

Algorithm 2: Centralized VM allocation algorithm

```
L = sorted list of new VMs (arriving at time t) by
decreasing volume
S = sorted list of incomplete servers in all DCs by
increasing size of available number of vCPU
for all l ∈ L do
    find list S_opt of servers with minimum volume v
    find server s ∈ S_opt with minimum cost:
    min_{1≤i≤M} Ec_i(P'_i + P_idle + c_L[0] * P_c + E_ON) − EP'_i
    allocate l on s
```

Each VM is characterized by its execution time, its CPU requirement and its memory usage. A given server can run a VM only if CPU and memory requirements of the VM are satisfied by this server. We define the *volume* of a VM as the product of these two values.

Algorithm 2 executes a Best-Fit algorithm based on the volume. More precisely, it orders VMs by decreasing volume and servers of all DCs by increasing free volume, and allocates VMs one by one on the server of minimum free volume that can execute it. If many servers with same free volume can execute the VM, the server with minimum expected brown consumption cost is selected. As this expected cost is the same for two servers of the same DC (homogeneous servers), this implies to select the best location among DCs with servers of minimum free volume. This is especially the case when no incomplete server can run the VM and a new server needs to be turned on.

The complexity of this greedy algorithm is in worst case $O(n_w \times n_S + n_w \log(n_w) + n_S \log(n_S))$ with n_w the number of waiting VMs and n_S the number of servers over all DCs. It corresponds to a worst case where, for all waiting VMs and all server on, the free volume of the server is higher than the volume of the task but it cannot run it.

The second allocation step is executed by Algorithm 3. This second pass permit to lowered the possible negative impact of the Best-Fit algorithm that mostly focused on volume issues.

Algorithm 3: Allocation Revision.

```
Order DC by increasing ERGE
Let Lt_i the list of pre-allocated VMs on incomplete
servers of DC_i
for all DC_sending in {1..M-1} do
  DC_receiving = M
  Order Lt_DC_sending by decreasing volume
  while DC_sending < DC_receiving AND Lt_DC_sending not
  empty do
    for all VM t ∈ Lt_DC_sending do
      if a server of DC_receiving can run t AND
      Exp_Cost_with_Migration <
      EP'_DC_sending + EP'_DC_receiving then
        Pre-allocate t on DC_receiving
        Remove t from Lt_DC_sending
    DC_receiving − −
```

Algorithm 3 first orders DCs by expected remaining green energy (ERGE). More precisely, it evaluates the expected green energy not used before the night. This step permits to determine the DCs that will send VMs in this process and those that will receive new ones. VMs pre-allocated on DCs with lower ERGE will be moved to DCs with higher ERGE. The evaluation of ERGE for each DC is done at maximum over n_{eval} time slots, as detailed in Subsection IV-A. In increasing order of ERGE, DCs consider the possibility to move their pre-allocated VMs to greener DCs by decreasing order, and execute a Best-Fit algorithm. Algorithm 3 focuses on VMs allocated on incomplete servers, that is, servers that could be profitably turned off in the consolidation process. Each VM is moved if it reduces the Cloud's overall brown energy consumption, based on our evaluation.

Algorithm 3 has a complexity in $O(\sum Npa_i \log(Npa_i))$ with Npa_i the size of Lt_i. Its complexity is thus of lower order than the complexity of Algorithm 2.

C. VM migration

The following step of NEMESIS determines VMs migrations inter DCs. This step consists in moving running VMs to allocate them on other DCs with the objective to reduce the brown energy consumption.

Due to bandwidth constraints, a DC can only migrate VMs one by one. Thus, the amount of VMs a DC can send in a single time slot is bounded by the sum of migration times. In addition, both DCs have to be synchronized during the migration time. Thus, if many DCs send VMs to the same DC, they can not communicate during the same time.

Algorithm 5 first lists the VMs to migrate for each DC, denoted $List\_to\_migrate_i$. For this step, we first order VMs running on DC_i by decreasing remaining execution time. The first VMs are added to the list until the sum of migration times reaches the duration of the time slot. As migration are done one by one, there is no need to consider more VMs for migration. As in Algorithm 3, the DCs with lower ERGE will send VMs to DCs with higher ERGE. For given receiving and sending DCs, VMs are ordered by decreasing volume and allocated one by one if some servers can host them, if the expected brown power consumption is reduced and if the migration time constraints are fulfilled as illustrated in the example of Figure 1.

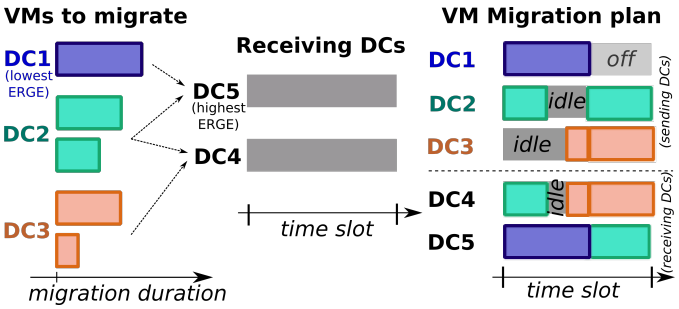

Fig. 1: Example of VM migrations with 5 DCs

To simplify the synchronizations, we limit each DC to send VMs to only two DCs during a time slot, or to receive VMs from only two DCs. For given receiving and sending DCs, we determine the VMs to migrate and the migration date. Migrations are done consecutively in the same time interval. The pair of sending and receiving DCs are determined as follow. First, worst DC DC_1 sends VMs to the best DC DC_M. Then, if some VMs remain to migrate, DC_1 sends VMs to DC_{M-1}. In the following, if last migration from DC_i is done to DC_j, then VMs from DC_{i+1} will be migrated first to DC_j, and then to DC_{j-1}.

Algorithm 4: Compute t_{limit} and t_{send} (for Algorithm 5)

```
t_limit = t_send;
forward = 1 − forward;
if forward = 1 then
  t_send = 0;
else
  t_send = Δt;
```

For given receiving and sending DCs, the time interval is determined by variables t_{limit} and $forward$. If $forward$ equals 1, tasks migrations are scheduled between time 0 and t_{limit}, and between t_{limit} and Δt in the other case. Then, t_{send} corresponds to the bound for the next task migration. If $forward$ equals 1, first task migration occurs at time 0 and t_{send} is the time of the end of the last scheduled migration.

If $forward$ equals 0, we start at the end of the time slot. The first allocation of a migration is scheduled to finish at time Δt and t_{send} corresponds to the beginning of the last scheduled migration.

Algorithm 5: Migration algorithm

```
1: Order DC by increasing ERGE
2: Create lists List_to_migrate
3: DC_receiving = M; DC_sending = 1; n_receive = 0
4: t_send = 0; t_limit = Δt; forward = 1;
5: while DC_sending < DC_receiving do
6:    Order List_to_migrate_{DC_sending} by decreasing volume;
7:    Compute t_limit and t_send;
8:    Determine migration decisions;
9:    if migration decision then
10:      n_receive + +;
11:   if List_to_migrate_{DC_sending} is not empty then
12:      DC_receiving − −;
13:      n_receive = 0;
14:      Compute t_limit and t_send;
15:      Determine migration decisions;
16:      if migration decision then
17:         n_receive + +;
18:   DC_sending + +;
19:   if n_receive = 2 then
20:      DC_receiving − −; n_receive = 0;
```

Finally, Algorithm 5 is designed as follows. It first orders DCs by increasing ERGE, and creates a list of VMs to migrate. Then, it considers DCs one by one in increasing order of ERGE (from the minimum green energy available to the maximum). For each DC, it considers a first DC where to send VMs. Then, in decreasing order of VM volume, it determines if the migration is possible (if a server can run it and if migration time is available) and preferable (in terms of brown power consumption). After all VMs of the list are examined, the same is done with a second receiving DC. This is done until all DCs have been considered for either sending or receiving. Algorithm 5 has a complexity in $O(\sum Nr_i \log(Nr_i))$ with Nr_i the number VMs running on DC_i.

D. Consolidation

In the consolidation algorithm, Algorithm 6, the objective is to turn off the maximum possible servers through migrations intra DC. This algorithm executes independently on each DC. It first orders servers by decreasing empty volume. It then tests if the first servers can be turned off. In order to determine the number of servers to turn on, it executes a dichotomy on the number of servers. Then for any number of servers, it selects all VMs on these first servers, and tries to allocate it by a Best-Fit algorithm on the remaining servers. For $|L|$ incomplete servers, it first tries to turn off $k = \lfloor |L|/2 \rfloor$ servers. Then, k is increased if a solution exists and decreased in the other case. The algorithm continues until the optimal decision is decided.

Algorithm 6: Consolidation on each DC

```
Order list L of incomplete servers by decreasing empty volume
k = ⌊|L|/2⌋; i = |L|; sol = true
while i >= 2 do
   i = ⌈i/2⌉
   if sol then
      k+ = i
   else
      k− = i
   if VMs on the k first servers of L can be allocated on the |L| − k last ones then
      sol = true
   else
      sol = false
if sol=false then
   k − −
Allocate VMs on the k first servers of L on the |L| − k last ones
Turn off the k first servers
```

Algorithm 6 has a complexity in $O(Ns_i \log(Ns_i) + \log(Ns_i)(Nr_i \log(Nr_i) + Nr_i Ns_i))$ with Ns_i the number of incomplete servers of DC_i. Finally, NEMESIS has a simplified coarse-grain complexity in $O(n_S \times n_{VM} \log(n_w \times n_{VM})$ with n_{VM} the number of running and waiting VMs at current time slot, and n_S the number of servers on. As shown in this section, this value is a large overestimate of the real complexity.

V. Validation

A. Experimental setup

We evaluate the performance of NEMESIS through a modeling an simulation process. We use the SimGrid simulation framework [3] to perform our experiments with real-world IaaS Cloud and PV traces. SimGrid is a versatile platform (i.e. it can be used to study cluster, grid, peer-to-peer, Cloud, wide/local-area networks, ...) dedicated to the scalable simulations of distributed applications and platforms. It is grounded on sound simulation models (of CPUs, TCP/IP networks, VMs, and energy consumption) which are theoretically and experimentally assessed by numerous scientific works. Using these models, SimGrid accurately simulates the resources usage (i.e. CPU and bandwidth sharing), the execution time and the energy consumption of a distributed application code – NEMESIS in our case. In our experiments, we study a Cloud infrastructure with characteristics equivalent to the French experimental testbed Grid'5000 [13]. Thus, our Cloud comprises a total of 1,035 servers spread across 9 geographically distributed DCs. The DCs are linked together thanks to 10 Gigabit IP links. Inside each DC, the servers use 1 Gigabit Ethernet links.

The characteristics of each server are based on the Taurus servers of Grid'5000. These servers are equipped with 2 Intel

Xeon E5-2630 CPU with 6 cores each, 32GB memory, 598GB storage. To determine the power consumption of each node, we implement the power model of [9] that is based on real measurements made on Taurus nodes. These measurements notably state that a Taurus server consumes 8W when powered OFF, 97W when idle, and 220W at 100% CPU load. Moreover, with this model a server consumes 200W during 10 seconds when powering off, and 127W during 150 seconds when powering on.

We use the Eucalyptus IaaS Cloud traces of [14] that combines the traces of six different real production systems. It consists in the list of VM arrival times, required numbers of cores and execution lengths. We assume all the VMs to require 2Go of RAM per required core (following the typical characteristics of T2 VM instances of Amazon EC2 [15]). This workload trace spans over one week. We scale the workload to 80% of the total computing capacity of the Cloud. For the VMs to be always deployable in our Cloud, we limit the maximum number of cores a VM can requires to 6. The simulated workload is not a favorable scenario for on-site PV production as it does not follow a day/night pattern with reduced workload during nights when green production is null. Such an unfavorable workload is typical of IaaS Clouds [14] and thus, represents a realistic scenario. As these Cloud traces do not contain the CPU utilization over time of each VM, we assume that during its entire duration, each VM uses its allocated CPU resources at maximum. This constitutes the least favorable case from an energy point-of-view as it is the most consuming scenario.

We use real recordings of green power production collected by the Photovolta project [16] of the University of Nantes. These data correspond to the power produced by a single Sanyo HIP-240-HDE4 PV panel (with a total power of 5,52 Wc and a surface of $5.52m^2$) updated every five minutes over one week. In order to have heterogeneous trajectories between DCs (and thus to represent solar irradiance differences between sites spread across a country), we select recordings starting at different dates. For each DC, the number of PV panels is dimensioned to have one PV for 3 servers and PV signals are scaled accordingly.

The controller implementing NEMESIS runs at each time slot of five minutes. It records all the amounts received from the green power sources during the current day. The controller computes at each time slot the standard deviations $p_i(t)$ using this history. Following our approach in [11], our implementation of NEMESIS computes each expected green power production $Eg_i(t)$ by averaging a reference green power production trajectory (the Photovolta project recording of the 20th August 2013 in our case which is the day with the best yield) scaled according to the last green power production received from i.

We compare NEMESIS performance against four approaches of the literature:

- **Round-Robin** distributes the VMs fairly among the DCs regardless of their green power production;
- **First-Fit** deploys each VM on the first (according to an arbitrary predefined order) DC which can host it.
- **Modified Best Fit Decreasing (MBFD)** [17] is a reference approach for reducing power consumption in Clouds. It relies on a decreasing best-fit algorithm to allocate incoming VMs and to perform consolidation of running VMs. The consolidation consists in performing live migrations of VMs that run on underused servers (i.e. servers that have less than 50% of CPU used) and shutting down these servers to save energy. On the contrary to NEMESIS, MBFD does not take into account the network constraints and the local green power productions. The remaining execution time of the VMs and the energy cost of the live migration are also not considered when migrating a VM.
- **OOD-MARE** [18] is another approach from literature consisting in allocating the incoming and running VMs according to the current local green power productions. With this approach, a Most Available Renewable Energy (MARE) algorithm deploys the VMs on the DC that has the highest amount of available green energy. According to an Optimal Online Deterministic (OOD) policy, the running VMs that start consuming brown energy are sequentially re-allocated using live migrations. On the contrary to NEMESIS, OOD-MARE does not rely on green production forecasts and does not perform intra-DC consolidation. It also does not consider the VM remaining execution time when performing live migration. Finally, the energy cost of these live migrations is neglected.

Like NEMESIS, these algorithms firstly employ servers already ON, and switch OFF those who are idle. As the performance of First-Fit and MBFD strongly depends on the order of the DCs, we test two opposite configurations corresponding to the best and the worst possible contexts. To define these contexts, we sort DCs according to the total amount of on-site green energy they provide. The best (resp. worst) context corresponds to the case where the PV traces are sorted in a decreasing (resp. increasing) order.

B. Results analysis

We simulate the Cloud behavior over one week. Table I shows the total cumulative energy consumption. We can see that NEMESIS consumes significantly less brown energy than the other approaches: at least 3.18%, and 13.26% maximum. It also slightly reduces the overall (i.e. brown and green) energy consumption.

As they do not take account of the local energy productions, First-Fit and MBFD have very variable performance depending on the scenarios considered : between the worst and the best scenarios the brown energy consumption differs of around 1 MWh. It is worth noting that MBFD has the highest overall energy consumption despite being designed to save energy. This is caused by its consolidation strategy that performs too many VM live migrations and neglects their energy cost. Thus, the energy consumed by the live migrations is more important than the energy saved by powering OFF the servers. Moreover,

as MBFD does not take into account network constraints, it performs too many simultaneous migrations. These migrations compete for the network bandwidth, and take then significantly longer to terminate. As a consequence, the energy costs due to migrations is increased.

TABLE I: Total cumulative energy consumption in the best/worst contexts (if different). The differences with NEMESIS are in parenthesis.

Approaches		Overall consumption	Brown Consumption
NEMESIS		17.6 MWh	11.6 MWh
OOD-MARE		17.9 MWh (1.6%)	12 MWh (3.2%)
MBFD	**best**	18 MWh (1.9%)	12.2 MWh (4.47%)
	worst		13.3 MWh (13.9%)
First-Fit	**best**	17.7 MWh (0.7%)	12 MWh (3.4%)
	worst		13.2 MWh (13.3%)
Round-Robin		17.8 MWh (1%)	12.4 MWh (6.9%)

Figure 2 shows the simulated power consumption over time of each DC in the Cloud using NEMESIS. As expected, NEMESIS uses in priority the DC that has the highest power production (e.g. Grenoble and Rennes DCs). We can also observe that the consolidations significantly lower the power consumption of the concerned DC. For instance in the DC of Rennes, a consolidation of 22 VMs occurs at time 344,100 sec and lower the consumption of about 500 W.

Figure 3 shows the contributions of the three main algorithms constituting NEMESIS: Algorithms 3 (migration of pre-allocated VMs), 5 (inter-DC migration of running VMs) and 6 (consolidation). We measure the improvement given by an algorithm by disabling it and checking the resulting increase in the total cumulative brown energy consumption. We can see that the energy consumption (both brown and global) is mainly reduced by live migrations of VMs to other DCs (i.e. Algorithm 5). The two other algorithms have only a limited positive impact on the energy consumption. This simulation-based validation shows promising results considering it employs a non-favorable workload for solar energy production (i.e. not exhibiting day/night patterns).

C. Comparison with a theoretical lower bound

Considering that our scheduling problem is NP-hard, an optimal off-line solution cannot be computed in reasonable time. However, to provide indirect guarantees, we compute a theoretical (i.e. unreachable) lower bound for the total and brown energy consumption. To compute this lower bound, we consider a simpler problem where all the servers and energy productions of the Cloud are grouped in a single DC. We also neglect the delays and energy cost of VM migrations. Each time a VM starts or stops, we reallocate all the running VM to the servers using a best-fit decreasing algorithm. We employ in priority the servers that are powered ON, and we switch off all the idle servers. We found a theoretical lower bound of 10.62 MWh (resp. 17.2 MWh) for the brown (resp. total) energy consumption. The brown power consumption of NEMESIS is then 9.56% above this unreachable lower bound. This experiment demonstrates that the brown power consumption of NEMESIS is less than 9.56% above the optimal solution.

VI. Conclusion and future work

In this paper, we propose NEMESIS, an energy-efficient IaaS resource management framework to cope with distributed Cloud infrastructures with on-site PV production. As major Cloud providers resort more and more to solar energy, practical and realistic solutions are needed to optimize both the green and the brown consumption of DCs.

Our approach combines greedy resource allocation, inter DC network-aware live-migration, intra DC VM consolidation and PV production prediction algorithms. To the best of our knowledge regarding the state of the art, our work differs from the literature on the following points: 1) we deal with network constraints when performing VM migrations, 2) we rely on an original stochastic model to predict brown energy production, 3) we take into account on/off switching costs and delays.

In the simulation-based evaluation, we consider a realistic context as we use (1) real workload trace, (2) existing cloud infrastructure, (3) real photovoltaic power productions traces, and (4) an energy consumption model that is based on real wattmeter measurements. Moreover, our simulations are grounded on the SimGrid models for CPU usage, TCP/IP networks and VM live migrations that have been validated by numerous scientific works. That is why, despite the inherent limits of simulation, we can put a high confidence on our simulation results.

Despite a very unfavorable context with a workload that does not follow a day-night pattern contrary to photovoltaic power productions, the results show that significant amounts of energy can be saved with NEMESIS compared to classical allocation and consolidation algorithms from literature. The gains become more important when comparing our solution with the algorithms currently deployed on real production systems —-i.e. round-robin consumes 6.9% more brown energy than our solution whereas first-fit consumes between 3.2% (in an ideal configuration) and 13.3% more brown energy. Other algorithms from the literature also perform well (i.e. OOD-MARE), but NEMESIS is able to save a bit more energy (the 1.6% and 3.2%). We also examine the contribution of each NEMESIS component to the overall energy gain.

Our future work includes dealing with heterogeneous servers and studying the scaling properties of NEMESIS by increasing the number of DCs. But, it will require to find real Cloud workload traces in order to further validate our framework and convince Cloud providers to implement it. We also considered only independent VMs in this work. We plan to extend our proposal to take account of multi-VM applications and their QoS constraints that may limit the VM deployment and migration in the Cloud.

Acknowledgments

This work has been supported by the Inria exploratory research project COSMIC (Coordinated Optimization of SMart grIds and Clouds).

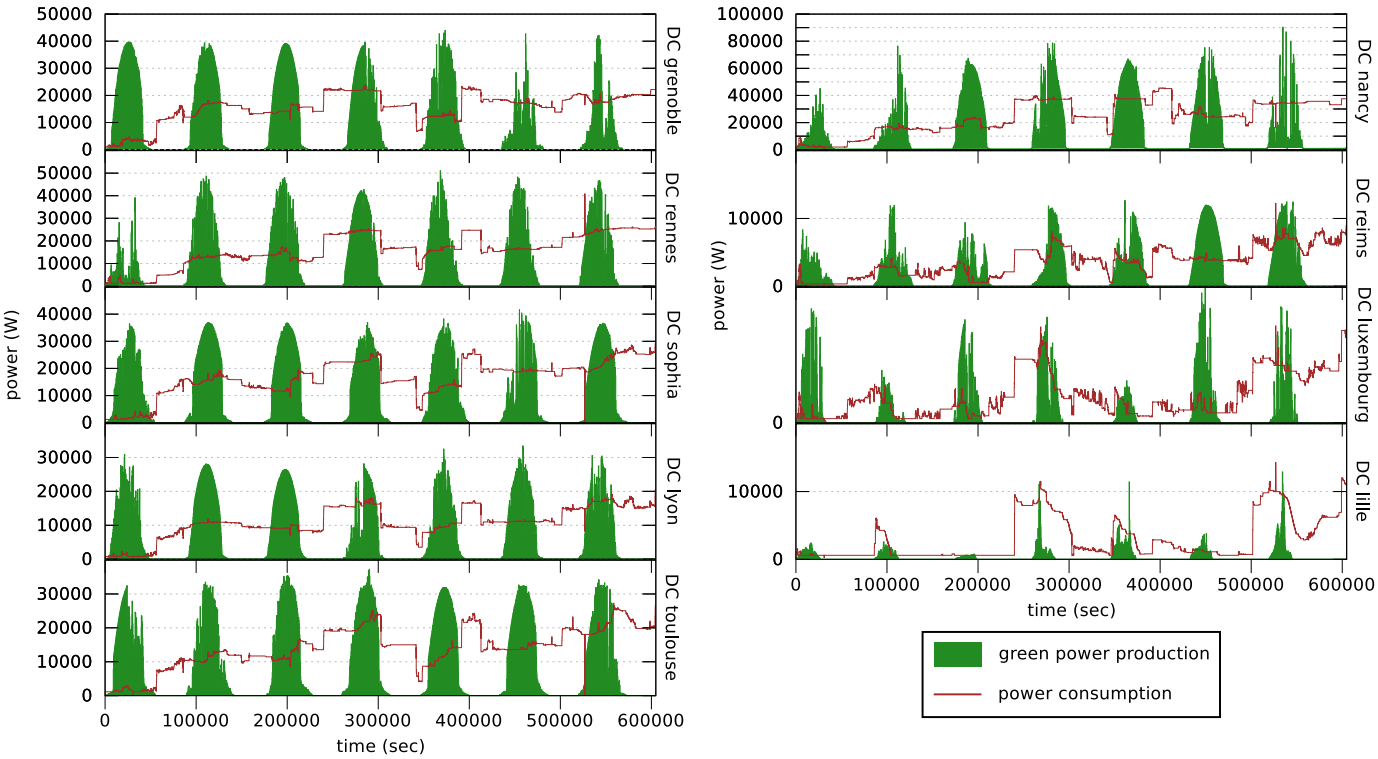

Fig. 2: Power consumption per DC with NEMESIS.

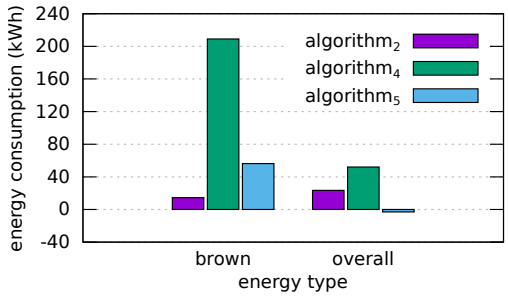

Fig. 3: Decrease in the energy consumption induced by the different algorithms composing NEMESIS.

REFERENCES

[1] M. Bertoncini *et al.*, "Next Generation Data Centers Business Models Enabling Multi-Resource Integration for Smart City Optimized Energy Efficiency," in *ACM e-Energy*, 2015, pp. 247–252.

[2] H. Goudarzi and M. Pedram, "Hierarchical SLA-Driven Resource Management for Peak Power-Aware and Energy-Efficient Operation of a Cloud Datacenter," *IEEE Transactions on Cloud Computing*, vol. 4, no. 2, pp. 222–236, 2016.

[3] H. Casanova *et al.*, "Versatile, scalable, and accurate simulation of distributed applications and platforms," *J. of Par. and Dist. Comp.*, vol. 74, no. 10, pp. 2899–2917, 2014.

[4] "Clicking Green: who is winning the race to build a green Internet," Greenpeace report, 2017.

[5] C. Chen, B. He, and X. Tang, "Green-aware workload scheduling in geographically distributed data centers," in *IEEE CloudCom*, 2012.

[6] A. Rahman, X. Liu, and F. Kong, "A Survey on Geographic Load Balancing Based Data Center Power Management in the Smart Grid Environment," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 214–233, 2014.

[7] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Toward energy-efficient cloud computing: Prediction, consolidation, and overcommitment," *IEEE Network*, vol. 29, no. 2, pp. 56–61, 2015.

[8] Y. Lin and H. Shen, "EAFR: An Energy-Efficient Adaptive File Replication System in Data-Intensive Clusters," *IEEE Trans. on Par. and Dist. Systems*, vol. 28, no. 4, 2017.

[9] Y. Li, A.-C. Orgerie, and J.-M. Menaud, "Opportunistic Scheduling in Clouds Partially Powered by Green Energy," in *IEEE GreenCom*, 2015.

[10] P. Velho, L. Schnorr, H. Casanova, and A. Legrand, "On the Validity of Flow-level TCP Network Models for Grid and Cloud Simulations," *ACM Transactions on Modeling and Computer Simulation*, vol. 23, no. 4, 2013.

[11] B. Camus, F. Dufossé, and A.-C. Orgerie, "A stochastic approach for optimizing green energy consumption in distributed clouds," in *SMARTGREENS*, 2017, pp. 47–59.

[12] E. Lorenz, J. Hurka, D. Heinemann, and H. G. Beyer, "Irradiance forecasting for the power prediction of grid-connected photovoltaic systems," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 2, no. 1, pp. 2–10, March 2009.

[13] "Grid'5000," https://www.grid5000.fr.

[14] R. Wolski and J. Brevik, "Using parametric models to represent private cloud workloads," *IEEE Transactions on Services Computing*, vol. 7, no. 4, pp. 714–725, 2014.

[15] "Amazon ec2 instances," https://aws.amazon.com/ec2/instance-types/.

[16] "Photovolta project," http://photovolta2.univ-nantes.fr.

[17] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing," *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755 – 768, 2012.

[18] A. Khosravi, A. N. Toosi, and R. Buyya, "Online virtual machine migration for renewable energy usage maximization in geographically distributed cloud data centers," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 18, 2017.