Green energy aware scheduling problem in virtualized datacenters

Gilles Madi-Wamba*, Yunbo Li*†, Anne-Cécile Orgerie†, Nicolas Beldiceanu* and Jean-Marc Menaud*
*IMT Atlantique, LS2N, Nantes, France – Email: {gmadiw14, nicolas.beldiceanu, jean-marc.menaud}@imt-atlantique.fr
†CNRS, IRISA, Rennes, France – Email: {yunbo.li, anne-cecile.orgerie}@irisa.fr

***Abstract*—With the generalization of cloud infrastructures usage, energy consumption has become a major issue. Scheduling heuristics have been proposed to optimize the resource usage of data center so as to take down the energy consumption. This paper tackles the problem with a different approach by taking into consideration the availability of renewable energy. First we formalize the green energy aware scheduling problem (GEASP) and propose a global model based on constraint programming as well as a search heuristic to solve it efficiently. The proposed model integrates the various aspects inherent to the dynamic planning in a data center: heterogeneous physical machines, various application types (i.e., active or online applications and batch applications), actions and energetic costs of turning ON/OFF physical machines, interrupting/resuming batch applications, CPU and RAM resource consumption, tasks migration, migration costs, and integration of green energy availability. The model can therefore reduce both the costs related to energy consumption and the carbon footprint of a data center. We evaluate the model against the state-of-the-art framework PIKA on real-world workload and solar power traces.**

I. Introduction and related work

Over the years, energy consumption has become a major concern in the field of information technologies. Amazon reports that the cost related to the energy consumption over a 3-years period is more than 40% of the overall cost of its data center [11]. Besides, in [3], Barrosso shows that over the lifetime of a data center, the expenses related to the energy consumption can easily surpass the hardware cost. In the literature, most work that attempts to reduce the energetic cost of a data center proceeds by reducing the overall energy consumption [9], [8]. Apart from the economic aspect, massive energy consumption also has repercussion on the environment, as the brown energy is produced from polluting sources. In this paper, we propose not only to reduce the brown energy consumption to handle the economic issue, but also to maximize green energy consumption to take care of the environmental issue.

The energy consumption of a system comprises a static and a dynamic part [16]. The static part is linked to the system size and the hardware type, and the dynamic part is linked to resources usage. In [15] and [6], Li et al. propose a novel framework, oPportunistic schedullIng broKer infrAstructure (PIKA) that aims at reducing the brown energy consumption of a small/medium-size data center. According to the amount of green energy available at each time slot PIKA turns physical machines ON/OFF, pauses, resumes or migrates jobs. However, this opportunistic scheduling is rather reactive than proactive. In this paper, we first formalize the green energy aware scheduling problem (GEASP). Second, we propose a global constraint programming model as well as a search heuristic and show that our model can significantly optimize the overall energy consumption of a data center. The proposed model simultaneously takes into account each of the following aspects:

- compatibility with both active and batch application types,
- applications resources requirements varying over the time,
- migration of applications from one physical server to another,
- energetic cost of application migrations,
- interruption/resumption of batch applications,
- compatibility with heterogeneous physical servers i.e. the resources are variable from one physical server to another,
- turning ON/OFF the physical servers and resources management,
- extra energy consumption to turn a physical server ON/OFF,
- green energy integration.

The main advantages of constraints programming include a certain level of flexibility that PIKA lacks. Indeed, thanks to this feature, the proposed model is also compatible with business constraints inherent to data center infrastructure management as well as with user preferences. Figure 1 illustrates different aspects of this context.

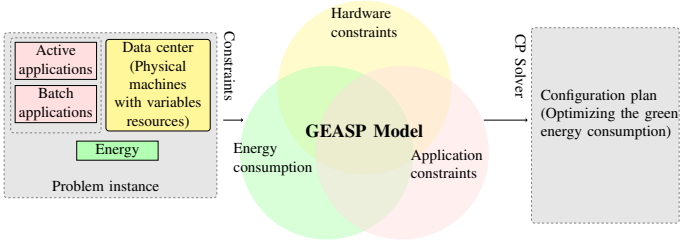


Fig. 1: The GEASP model computes a global and pro-active configuration plan that minimizes the brown energy and maximizes the green energy consumption of the data center while satisfying all the constraints. Hardware constraints are related to memory/CPU limitations and turning the servers ON and OFF. The application constraints are related to the interruption, the migration, and the placement of the applications as well as to some business constraints (ban, lonely, spread, ...).

The rest of the paper is organized as follows: Section II describes the problem and gives all the necessary definitions and Section III formalizes the GEASP problem and presents our model. In Section IV, we present the search heuristic to find suitable solutions to the GEASP. Finally, we evaluate our contribution using real data center workloads and green energy traces in Section V, and we conclude in Section VI.

II. Problem description and definitions

We consider the problem of scheduling in a small/medium size data center with limited resources [5]. The scheduling should minimize the brown energy consumption and maximize the green energy consumption. We recall that a data center is made up of one or more servers that host the applications. We have two types of applications, the active and the batch. Both types of application are introduced in Section II-A.

A. Definitions

Active Application An *active application* is an application that is executed throughout the time horizon without being interrupted. Each active application is described by its:

- Memory consumption, i.e. is the constant amount of RAM consumed by the application within the server it is hosted.
- CPU Usage, the processing power used by a server to execute the application; it may vary from one time slot to the next throughout the horizon.
- Migration cost, i.e. the extra energy consumption needed to migrate an application from one server to another.

An active application a_i of memory requirement $\mathbf{mem_i}$, CPU usage $\mathbf{cpu_i}$ and migration cost $\mathbf{migr_i}$ is denoted $\mathbf{a_i}(\mathbf{mem_i}, \mathbf{cpu_i}, \mathbf{migr_i})$.

Batch Application A *batch application* or *batch job* is an application that can be interrupted and resumed. A time slot during which a batch job is running is called a runtime. Each batch application is characterized by its :

- Memory consumption: its memory consumption of a batch job is constant.
- CPU Usage: CPU usage of a batch job is constant.
- Duration.
- Migration cost.
- Slack. The total amount of time during which it is interrupted shall not exceed a given slack.

A batch job b_i of memory $\mathbf{mem_i}$, CPU usage $\mathbf{cpu_i}$, duration d_i, migration cost $\mathbf{migr_i}$ and slack s_i is denoted $\mathbf{b_i}(\mathbf{mem_i}, \mathbf{cpu_i}, \mathbf{d_i}, \mathbf{migr_i}, \mathbf{s_i})$.

Server We define a *server* in this problem as a machine that hosts the applications. It is characterized by:

- its Memory limit is the total amount of memory that is shared among the hosted applications.
- its CPU limit is the total amount of processing power that can be shared among hosted applications.
- its ON (resp. OFF) energy is the constant amount of energy required to turn the server ON (resp. OFF).
- its load/consumption relation, which relates the CPU load of the server to its energy consumption.
- its id is a unique integer that identifies each server.

A server i of memory $\mathbf{MEM_i}$, CPU limit $\mathbf{CPU_i}$, ON energy $\mathbf{E_{on_i}}$, OFF energy $\mathbf{E_{off_i}}$, load/consumption relation r and id id_i is denoted $\mathbf{s_i}(\mathbf{MEM_i}, \mathbf{CPU_i}, \mathbf{E_{on_i}}, \mathbf{E_{off_i}}, \mathbf{r_i}, \mathbf{id_i})$.

Our goal is to compute an energy efficient configuration plan for the data center which, while reducing the amount of brown energy used and increasing the proportion of green energy, copes with the operational and the business constraints of different application types. Next definition introduces the notion of *Configuration plan*.

Configuration plan Given a time horizon, a set of active and batch applications and a data center, a configuration plan is a scheduling that meets the following requirements:

- At any time slot through out the horizon, it specifies for each active application an allocated server. A migration occurs when the host of an active application changes from one time slot to the next.
- It specifies when each batch job starts, when it ends, and eventual interruption and resumption times. For any batch job, the configuration plan specifies the hosting server at any timepoint. A migration occurs when the host changes from one time slot to the next.
- For each server of the data center, it specifies every time slot when the server is turned ON or OFF.

The configuration plan is energy efficient when it maximizes the green energy consumption of the data center. Indeed, the availability of green energy fluctuates over the time. Typical example is the solar energy.

Section III formalizes this problem into a constraint model.

III. Model of the problem

In this section, we propose a constraint based model to capture the already mentioned aspects of the green energy aware scheduling problem in virtualized data centers. Before detailing different parts of the model, we first shortly recall key notions of constraint programming in the context of virtualized data centers.

A. Background on constraint satisfaction problems

Constraint satisfaction problem A constraint satisfaction problem (CSP) [17] is a triple $P = (X, D, C)$ where :

- X is an tuple of $n > 0$ variables, $X = \langle x_0, x_1, \ldots, x_{n-1} \rangle$.
- D is a tuple of $n > 0$ domains, $D = \langle D_0, D_1, \ldots, D_{n-1} \rangle$ such that $x_i \in D_i$.
- C is a tuple of $k > 0$ constraints $C = \langle C_0, C_1, ..., C_{k-1} \rangle$.

We now define the notion of constraint.

Constrain Given a CSP $P = (X, D, C)$, a *constraint* $C_i \in C$ is a pair $\langle R_i, S_i \rangle$ where S_i is a subset of X, and R_i is a relation over S_i. The relation R_i specifies the tuples of values forbidden among the Cartesian product of the domains of the variables in S_i.

Solution of a CSP Given a CSP $P = (X, D, C)$, a *solution* S of P is a tuple $\langle v_0, v_1, ..., v_n \rangle$ such that $v_i \in D_i$ and for every constraint $C_i = \langle R_i, S_i \rangle \in C$, the relation R_i holds.

B. Constraint programming for the green energy aware scheduling problem in virtualized data center

Constraint programming is a suitable solution to solve most scheduling problems [2]. In the case of a green energy aware scheduling problem, constraint programming is well suited as it has a good expressiveness to precisely model each component of the problem. It also enables to add any new constraint to an existing model. In [13], Hermenier et al. present four side constraints that may be required by the administrator of a system or by the users to restrict the placement of the applications into servers.

- **Ban**. At a given time t, the system administrator may want to turn off a server to perform a hardware or a software maintenance.
- **Spread**. In order to achieve tolerance to hardware failures, an application may use replication. For the fault tolerance, the application and its replication should run at any time on different servers.
- **Lonely**. For some reasons, an application may require to run alone in a server.
- **Capacity**. Sometimes, due to some shared resources, it might be useful to limit the number of applications executed in a server. The capacity constraint is used to ensure that the number of applications hosted on a given server is below a given limit.

This list of side constraints is not exhaustive. Besides the good expressiveness and the flexibility, the solving process of a constraint satisfaction problem through constraint propagation [17] ensures that all the constraints of the problem are satisfied.

We now formalize the definition of a green energy aware scheduling problem.

Green energy aware scheduling problem A *green energy aware scheduling problem* (GEASP) is a tuple such that $\mathbf{P} = \{\mathbf{S}, \mathbf{A}, \mathbf{B}, \mathbf{T}, \mathbf{E}\}$ where:

- S is a set of servers.
- A is a set of active applications.
- B is a set of batch jobs.
- T is the time horizon: an integer specifying the total number of time slots to consider.
- E is a time series of length T which specifies the quantity of green energy available at each time slot.

In Sections III-C up to III-G, we present how we formalize each aspect of a **GEASP** into a constraint satisfaction problem. We will use the toy example III.1 to illustrate our model.

Example III.1. Throughout this paper, we use the following toy **GEASP**. $P = \{S, A, B, T, E\}$ where:

- $S = \{s_0(3, 50, 3, 3, r, 0), s_1(3, 50, 3, 3, r, 0)\}$ where we have $r(cpu\_load) = 10 + cpu\_load$
- $A = \{a_o(1, 20, 2)\}$
- $B = \{b_0(2, 20, 3, 1, 5), b_1(2, 20, 3, 1, 5)\}$
- $T = 10h$
- $E = 10, 10, 10, 90, 90, 10, 10, 10, 48, 43$

An energy efficient configuration for the **GEASP** is given below:

- **Slot** 0 s_0 is turned ON, and a_0 is placed on s_0.
- **Slot** 1 No change.
- **Slot** 2 No change.
- **Slot** 3 b_0 is launched on s_0. s_1 is turned ON and b_1 is launched on it.
- **Slot** 4 No change.
- **Slot** 5 b_0 and b_1 are interrupted and s_1 is turned OFF.
- **Slot** 6 No change.
- **Slot** 7 No change.
- **Slot** 8 b_0 is resumed on s_0.
- **Slot** 9 b_1 is resumed and migrated from s_1 to s_0.

C. Turning a server ON or OFF

To model the action of turning a server ON and OFF, we have to consider two aspects. First we need to consider the inherent energetic costs of these actions, and second we need to ensure that no application can be scheduled on a switched off server.

1) Energetic cost: As seen in the definition of a server in Section II-A, each server has a turning ON/OFF cost. Let $P = \{S, A, B, T, E\}$ be a GEASP. For each server $\mathbf{S_i}(\mathbf{MEM_i}, \mathbf{CPU_i}, \mathbf{E_{on_i}}, \mathbf{E_{off_i}}, \mathbf{r_i}, \mathbf{id_i}) \in S$, a set of $T + 1$ boolean variables $ON_{(i,t)}$ indicate at each time slot $t(t \in [-1, T-1])$ if the server is ON or OFF. $ON_{(i,-1)} = \mathit{false}$ indicates that server S_i is initially off.

Turning a server ON or OFF at a time slot t consumes an extra energy at this time t. For each server $\mathbf{S_i}(\mathbf{MEM_i}, \mathbf{CPU_i}, \mathbf{E_{on_i}}, \mathbf{E_{off_i}}, \mathbf{r_i}, \mathbf{id_i})$, the model of this energetic cost relies on T switch task variables. Before we detail these variables, we first introduce the notion of task.

Task A task i is a tuple $task_i(s_i, d_i, e_i, < r_i >, h_i)$ where:

- s_i is the *start time of the task*.
- d_i is the *duration of the task*.
- e_i is the *end of the task*.
- $< r_i >$ is the list that contains all the *resource requirements* of the task. A task might consume one or more resources of different types.
- h is the server hosting the task.

Given a server $\mathbf{s_i}(\mathbf{MEM_i}, \mathbf{CPU_i}, \mathbf{E_{on_i}}, \mathbf{E_{off_i}}, \mathbf{r_i}, \mathbf{id_i})$, the energy consumption of its switch task at time t depends of the server s_i being turned ON or OFF at that time t. It is a variable $switch\_energy_{i,t}$ given by :

$$switch\_energy_{i,t} = E_{on_i} * (ON_{(i,t-1)} < ON_{(i,t)}) + E_{off_i} * (ON_{(i,t-1)} > ON_{(i,t)}). \quad (1)$$

where a term like $ON_{(i,t-1)} > ON_{(i,t)}$ is interpreted as a $0-1$ variable that will be assigned value 1 (resp. 0) if the corresponding conditions holds (resp. doesn't hold).

Example III.2. According to the configuration plan given in Example III.1 we have:

- $switch\_energy_{0,0} = 3$ Since server s_0 is turned ON at $t = 0$.
- $switch\_energy_{1,3} = 3$ Since server s_1 is turned ON at $t = 3$.
- $switch\_energy_{1,5} = 3$ Since server s_1 is turned OFF at $t = 5$.
- $switch\_energy_{0,t} = 0$ for all $t \neq 0$ and besides $switch\_energy_{1,t} = 0$ for all $t \in \{0, 1, 2, 4, 6, 7, 8, 9\}$

2) Availability of a server: The availability of a server is the constraint that makes a server unavailable for scheduling at any time slot during which it is switched off. Since any application consumes a strictly positive amount of memory on the server on which it runs, a server can not host any application if it has no free memory available. Thus to make a server unavailable, we set its available memory to 0. To do so, we create memory tasks whose aim is to fill any server's memory at any time t when the server is off.

In order to avoid creating T extra tasks, we add a memory resource $mem_{i,t}$ to the switch tasks of Section III-C1. The memory consumption $m_{i,t}$ of the switch task $\mathbf{switch_{(i,t)}}$ of a server $s_i(MEM_i, CPU_i, E_{on_i}, E_{off_i}, r_i, id_i)$ at time t is given by :

$$mem_{i,t} = MEM_i * (ON_{(i,t)} = 0). \tag{2}$$

The switch tasks of the model are therefore indexed by the server index i and the time slot t, they are denoted $\mathbf{switch_{(i,t)}}(\mathbf{t}, \mathbf{1}, \mathbf{t+1}, < \mathbf{energy_{i,t}}, \mathbf{mem_{i,t}} >, \mathbf{i})$.

Section III-D here after describes how we model active and batch applications using the notion of task introduced in Definition III-C1.

D. Modeling the applications

The most difficult part of a GEASP is the model of the applications. The model should take into consideration the following requirements:

- [Host]. At any time t any active application and any uninterrupted batch job should be hosted on a server.
- [Migration]. Both active and batch applications can be migrated from one server to another one. These migrations have costs that should be taken into consideration. Details on how to model the migration costs are given in Section III-E.
- Active applications are never interrupted.
- Batch jobs can be interrupted as long as the corresponding slack constraint is not violated.

In order to meet these requirements, we model each active application $\mathbf{a_i}(\mathbf{mem_i}, \mathbf{cpu_i}, \mathbf{migr_i})$ (resp. each batch application $\mathbf{b_i}(\mathbf{mem_i}, \mathbf{cpu_i}, \mathbf{d_i}, \mathbf{migr_i}, \mathbf{s_i})$) by a set S_{a_i} (resp. S_{b_i}) of tasks.

1) Active applications: As active applications are not interrupted, each active application $\mathbf{a_i}(\mathbf{mem_i}, \mathbf{cpu_i}, \mathbf{migr_i})$ is modeled by a set S_{a_i} of T tasks that are denoted $\mathbf{sub_{(a_i,t)}}(\mathbf{t}, \mathbf{1}, \mathbf{t+1}, < \mathbf{mem_{i,t}}, \mathbf{cpu_{i,t}}, \mathbf{energy_{i,t}} >, \mathbf{h_{i,t}})$, t from 0 to $T - 1$ where:

- $mem_{i,t}$ (resp. $cpu_{i,t}$) is the memory consumption (resp. the CPU usage) of the task $\mathbf{a_i}(\mathbf{mem_i}, \mathbf{cpu_i}, \mathbf{migr_i})$ at time t.
- $h_{i,t}$ is the server that hosts $\mathbf{a_i}(\mathbf{mem_i}, \mathbf{cpu_i}, \mathbf{migr_i})$ at time t.
- $energy_{i,t}$ is the amount of extra energy consumed by the server $h_{i,t}$ to host $sub_{(a_i,t)}$.

2) Batch applications: Since the duration of each batch job is fixed, any batch job $\mathbf{b_i}(\mathbf{mem_i}, \mathbf{cpu_i}, \mathbf{d_i}, \mathbf{migr_i}, \mathbf{s_i})$ is modeled by a set S_{b_i} of d_i tasks which are denoted $\mathbf{sub_{(b_i,j)}}(\mathbf{s_{(b_i,j)}}, \mathbf{1}, \mathbf{e_{(b_i,j)}}, < \mathbf{mem_{i,j}}, \mathbf{cpu_{i,j}}, \mathbf{energy_{i,j}} >, \mathbf{h_{i,j}})$ where $j \in [0, d_i - 1]$.

The batch applications can be interrupted and resumed. Therefore $s_{(b_i,j)}$ and $e_{(b_i,j)}$ are variables taking values in $[0, T - 1]$ with the following constraints:

$$\forall j \in [1, d_i - 1], s_{(b_i,j)} \leq e_{(b_i,j-1)} \tag{3}$$

The slack constraint is modeled as follows:

$$e_{(b_i,d_i-1)} - s_{(b_i,0)} - d_i \leq slack. \tag{4}$$

For both active and batch jobs, the variables $energy_{i,t}$ and $energy_{i,j}$ specify how much extra energy is needed by a server to host this given application. Its value varies from one host to another. Details on how this value is set are presented in Section III-G.

To ensure that a server s in which a task is scheduled at time t is ON at that time, we add the following constraint for each active and batch tasks.

$$\forall sub_{(a_i,t)}(t, 1, t{+}1, < mem_{i,t}, cpu_{i,t}, energy_{i,t} >, h_{i,t}) \in S_{a_i}, \quad h_{i,t} = s \Rightarrow ON_{(s,t)} = 1 \tag{5}$$

$$\forall sub_{(b_i,j)}(j, 1, j{+}1, < mem_{i,j}, cpu_{i,j}, energy_{i,j} >, h_{i,j}) \in S_{b_i}, \quad h_{i,j} = s \Rightarrow ON_{(s,t)} = 1 \tag{6}$$

E. Migration cost

This section models the migration cost of both active and batch applications. For any active application $\mathbf{a_i}(\mathbf{mem_i}, \mathbf{cpu_i}, \mathbf{migr_i})$ and any task $\mathbf{sub_{(a_i,t)}}(\mathbf{t}, \mathbf{1}, \mathbf{t+1}, < \mathbf{mem_{i,t}}, \mathbf{cpu_{i,t}}, \mathbf{energy_{i,t}} >, \mathbf{h_{(i,t)}}) \in S_{a_i}$, we create a migration task $\mathbf{migr_{(a_i,t)}}(\mathbf{t}, \mathbf{1}, \mathbf{t+1}, < \mathbf{migr\_energy_{i,t}} >, \mathbf{h_{(i,t)}})$ where $migr\_energy_{i,t}$ is the energy that is consumed whenever this active task a_i is migrated from one server to another at time t. Its value is given by:

$$migr\_energy_{i,t} = \begin{cases} 0 & t = 0 \\ migr_i * (h_{(i,t)} \neq h_{(i,t-1)}) & t > 0 \end{cases}$$

Similarly, we create migration tasks $\mathbf{migr_{(b_i,j)}}$ for batch applications.

Example III.3. According to the configuration plan given in Example III.1, we have:

- $migr\_energy_{b_1,9} = 1$ since batch application b_1 is migrated from server s_1 to server s_0 at slot 9.

F. Memory consumption and CPU usage

Each server has a limited quantity of memory and CPU. An application can be scheduled on a server $s_i \in S$ at time t if and only if server s_i has enough resources to satisfy the application requirements. To take these constraints into consideration, we use a *cumulative constraint* [1], [14].

Cumulative constraint Given a set T of single resource tasks, the cumulative constraint ensures that at any time t, the cumulative resource of the set of tasks scheduled at t does not exceed a fixed limit.

The standard definition of the cumulative constraint given above assumes that there is one single host for the tasks. We will instead use the *cumulatives constraint* [4], a slightly generalized version of the *cumulative constraint* which allows more than one host, i.e. more than one server. The *cumulatives constraint*:

`cumulatives(Tasks,Machines)`

where

- $Tasks$ is a set of single resource tasks of the form $task(start, duration, end, resource, host)$.
- $Machines$ is a set of fixed resource hosts of the form $machine(id, resource\_limit)$.

holds if, for each machine m, at each point t, where at least one task is assigned to machine m, the cumulative resource of the set of tasks assigned to m at t is less than or equal to (respectively greater than or equal to) the capacity associated with machine m.

The host of a task is the id of the server where it is scheduled. Before we detail the use of the cumulatives constraint in the case of memory and CPU usage, we first introduce the notion of restriction of a task to a given resource.

Restriction of a task Given a task $t(start, duration, end, < r_1, r_2, \dots r_n >, host)$ its restriction to the resource $r \in < r_1, r_2, \dots r_n >$ is the task $t\_r(start, duration, end, r, host)$.

To model the memory consumption and CPU usage of a GEASP, we will use two *cumulatives constraint*, $cumulatives(Memory\_Tasks, Memory\_Machines)$ and $cumulatives(CPU\_Tasks, CPU\_Machines)$ where:

- $Memory\_Tasks$ is the set of active tasks, batch tasks and switch tasks restricted to the memory resource.
- CPU_Tasks is the set of active tasks, batch tasks and switch tasks restricted to the CPU resource plus a set of complementary tasks that we use to compute the energy consumption of each server at each time slot. Details on these complementary tasks are given in Section III-G.
- $Memory\_Machines$ and $CPU\_Machines$ are the set of machines built in the following way:

  1 $Memory\_Machines$ = $\{machine\_mem(id_i, MEM_i)$ such that $\exists s_i(MEM_i, CPU_i, E_{on_i}, E_{off_i}, r_i, id_i) \in S\}$

  2 $CPU\_Machines$ = $\{machine\_cpu(id_i, CPU_i)$ such that $\exists s_i(MEM_i, CPU_i, E_{on_i}, E_{off_i}, r_i, id_i) \in S\}$

Example III.4. Figure 2 shows how the *cumulatives* constraint takes into consideration the RAM limit of each server at each time slot in the configuration plan given in Example III.1.

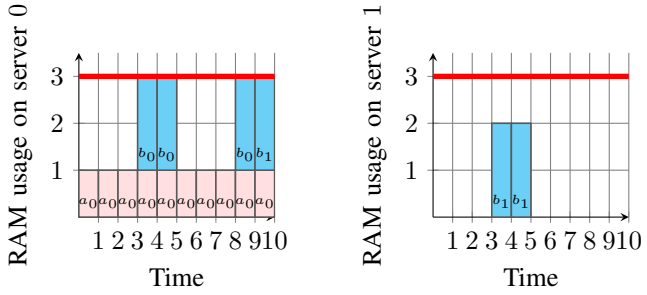


Fig. 2: *cumulatives* constraint for the RAM resource limits

G. Minimizing the brown energy consumption

Before we detail how to minimize the brown energy consumption of a data center, we first need to specify how to compute the energy consumption of a server. We assume server s_i is already turned ON at time $t \in [0, T-1]$ and there is no migration scheduled at time t. Then the energy consumption $e_{i,t}$ of s_i at time t is obtained from Equation (7) by:

$$e_{i,t} = r(load_{s_i,t}) \tag{7}$$

where $load_{s_i,t}$ is the CPU load in percent of s_i at time t. Therefore to know the energy consumption of a given active (resp. batch) task $\mathbf{sub_{(a_i,t)}(t, 1, t+1, < mem_{i,t}, cpu_{i,t}, energy_{i,t} >, h_{i,t})}$ (resp. $\mathbf{sub_{t,1,t+1,<mem_{i,j},cpu_{i,j},energy_{i,j}>,h_{i,j})}$) at time t, we first need to know the CPU capacity of the host $h_{i,t}$ (resp. $h_{i,j}$) in which it is scheduled at time t. The problem is that $h_{i,t}$ (resp. $h_{i,j}$) is not initially fixed, so we have no way to know in advance the CPU capacity of the host. To overcome this issue, we introduce the *element constraint* [18].

element constraint Given a list L of integer values, two integer variables $index$ and V, the $element(L, index, V)$ constraint ensures that V is the $index^{th}$ element of L i.e $V = L_{index}$.

Let $CPU\_Capacities$ be the list of the CPU capacities of the servers. The list is sorted in ascending order of servers id's. For any time $t \in [0, T-1]$ and for any active (resp. batch) task $\mathbf{sub_{(a_i,t)}(t, 1, t+1, < mem_{i,t}, cpu_{i,t}, energy_{i,t} >, h_{i,t})}$ (resp. $\mathbf{sub_{(b_j,j)}(s_{(b_i,j)}, 1, e_{(b_i,j)}, < mem_{i,j}, cpu_{i,j}, energy_{i,j} >, h_{i,j})}$) we state:

$$element(h, CPU\_Capacities, HostCapacity) \tag{8}$$

The energy consumption $energy$ of the task is therefore given by:

$$energy = r(\frac{cpu * 100}{HostCapacity}) \tag{9}$$

We now present the energetic model that we used in this paper to characterize the function r for all the servers.

Energetic model: As seen in Section II-A, each server is characterized by a relation r that relates its workload to its energy consumption. In [15], Li et al. experimented multiple tests on Taurus node at the Lyon site of Grid'5000, a large-scale and versatile test-bed for experiment driven research. Each node has 12 cores and each core presents 8.3% overall CPU utilization. They came out with the conclusion that, the overall energy consumption of the node depends on the number of activated cores. Table I summarizes these results.

# cores	0	1	2	3	4	5	6
Power (W)	97	128	150	158	165	171	177
# cores	7	8	9	10	11	12	
Power (W)	185	195	200	204	212	220	

TABLE I: Experimental energy consumption of a 12 cores node according to the number of activated cores

Following the idea of the PowerModel of CloudSim [7], we use Table I to implement the function *getPower()* in order to compute the energy consumption of a server. The pseudo code of *getPower()* is given in Algorithm 1. We thus have :

$$energy(h,t) = r(load(h,t)) = getPower(CPU_h, cpu\_used_{h,t}) \quad (10)$$

where CPU_h is the CPU capacity of host h and $cpu\_used_{h,t}$ is the amount of CPU used in host h at time t.

Algorithm 1 Procedure getPower() to compute the energy consumption of a server according to its CPU load

Require: $CPU\_Capacity$ ▷ The CPU capacity of the server
Require: CPU_Used ▷ CPU usage of the server
$Power\_table \leftarrow [97, 128, 150, 158, 165, 171, 177, 185, 195, 200, 204, 212, 220]$
$Step \leftarrow \frac{1}{12} \times CPU\_Capacity$
$A \leftarrow \frac{1}{Step+1} \times CPU\_Used$
$B \leftarrow \min(A+1, 13)$
$E_A \leftarrow Power\_table_A$
$E_B \leftarrow Power\_table_B$
$Energy \leftarrow E_A + ((E_B - E_A) \times CPU\_Used \times 10)/(CPU\_Capacity \times 10)$
return $Energy$

The value $energy(h,t)$ only takes into consideration the load of the host h. To obtain the actual consumption of the data center, we need to take into consideration all the extra energy costs resulting from turning the servers ON and OFF, and migrating the tasks over the different hosts.

Equation 11 states the total energy consumption of the data center at a given time slot t.

$$E_t = \sum_h energy(h,t) + \sum_a migr\_energy_{(a_i,t)} + \sum_b migr\_energy_{(b_i,t)} + \sum_{h,t} switch\_energy_{(h,t)} \quad (11)$$

where:

- $migr\_energy_{(a_i,t)}$ (resp. $migr\_energy_{(b_i,t)}$) is the extra energy consumed at time t whenever the active task a (respectively batch task b) is migrated from one host to another at time t.
- $switch\_energy_{(h,t)}$ is the extra energy consumed at time t when host h is switched ON or OFF at time t.

Equation 12 computes the total amount of **brown** energy consumed by the data center at time t.

$$B_t = \max(0, E_t - G_t) \quad (12)$$

where G_t is the amount of green energy available at time slot t. Finally, Equation 13 computes the total amount of **brown** energy consumed by the data center throughout the time horizon.

$$B = \sum_{t=0}^{T-1} B_t \quad (13)$$

Example III.5. Figure 3 shows the overall energy consumption of the data center with the configuration plan of Example III.1. As we can see from the plot, the batch jobs are run, migration are executed, and servers are turned ON preferably during time slots when the green energy is highly available. This is achieved by our heuristic that we present in Section IV.

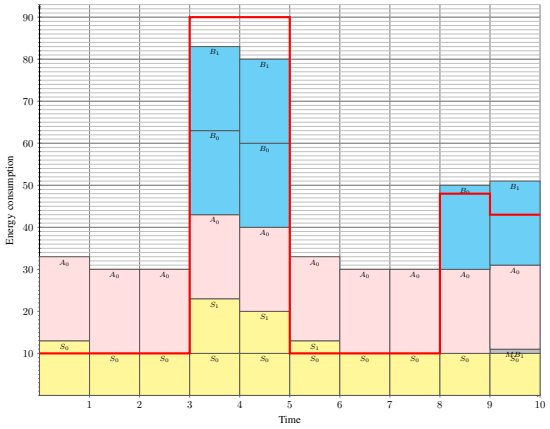


Fig. 3: Energy consumption of the configuration plan. The red curve is the green energy availability per slot

An energy efficient configuration plan is one that minimizes the total amount of brown energy B consumed by the data center throughout the horizon. Thus the overall objective of the GEASP is to minimize B.

IV. Solving the GEASP

According to the problem size, it might take a long time to find an optimal solution to a constraint satisfaction problem and to prove its optimality. To overcome this issue, it is important to have a dedicated search strategy that speeds up the process of finding a good solution.

We propose a dynamic heuristic that targets the starts of the batch tasks. Once these variables are fixed, the other variables of the problem will be fixed by propagation.

Since the batch tasks are the ones whose start and end variables are flexible, the intuition of the heuristic is to try to schedule them at times when the green energy is highly available. To do so, we proceed as follows.

1) Variables ordering: Let $b(mem, cpu, d, migr, slack)$ be a batch application modeled by the set S_b of batch tasks of the form $sub_{(b,i)}(s_{(b,i)}, 1, e_{(b,i)}, < mem, cpu, energy >, h)$. The order in which the start variables of the batch tasks in S_b are fixed is $\mathbf{s_{(b,0)}}, \mathbf{s_{(b,1)}}, \ldots, \mathbf{s_{(b,d-1)}}$. The purpose of this order is to take into consideration the precedence constraint stated in Equation 3 through out the search.

2) Values ordering: This part of the search strategies deals with the order in which values are chosen from the domain of each start variable. Let $s_{(b,i)}$ be the start of $sub_{(b,i)}$. We have $dom(s_{(b,i)}) = [0, T-1]$. Following the intuition, the idea is to try to fix $s_{(b,i)}$ to a value $t \in dom(s_{(b,i)})$ when the green energy availability is maximal. Doing this ensures that we maximize the green energy consumption. We proceed in two steps:

1 **Sort step**. This is the first step that is performed each time a start variable has to be instantiated. Let $E = e_0 e_1 \ldots e_{T-1}$ be the green energy of a GEASP where $E(t) = e_t, t \in [0, T-1]$ is the amount of green energy available at time t. We sort the values e_t in descending order and obtain the list $SortedE = e_{(0,t_0)}, e_{(1,t_1)}, \ldots, e_{(T-1,t_{T-1})}$ such that $\forall e_{(i,t_i)} \in SortedE, E(t_i) = e_{t_i} = e_{(i,t_i)}$
This gives the order in which values are chosen from $dom(s_{(b,i)})$, namely $t_0, t_1, \ldots, t_{T-1}$.

2 **Update step**. This step is performed immediately after a start variable is fixed. Let a batch task $sub_{(b,i)}(s_{(b,i)}, 1, e_{(b,i)}, < mem, cpu, energy >, h)$ and suppose $s_{(b,i)}$ is fixed to $t \in [0, T-1]$. Since running $sub_{(b,i)}$ consumes energy, this step updates the green energy available at time t. This update is done by decrementing $E(t)$ by $energy$. The following constraint updates the green energy.

$$E \leftarrow e_0, e_1, \ldots, e_t - energy, e_{t+1}, \ldots, e_{T-1} \qquad (14)$$

The *Sort step* and the *Update step* presented in this Section sketch the general idea of the heuristic to solve the GEASP. It is possible to optimize the *Sort step* by avoiding a total sorting over all the indices. The advantage of doing a partial sort is that, since some tasks are linked through a precedence relation, it does not delay too much the first tasks of the precedence chain.

V. Evaluation

To measure how effective is the model when used in a small/medium size data center, we run simulations on real workload traces.

As stated in [6], finding real traces for data center's activity is very difficult. For this, we studied anonymized traces provided by EasyVirt, a French SME monitoring Cloud data centers. We used 3 real workload data sets. Each data set reports the activity of a Cloud provider with 55 servers over a 12 hours period. The traces reports client requests for virtual machines with specified CPU and RAM needs. In the context of a virtualized data center, the virtual machines are used to encapsulate the applications that are executed on the physical machines. Before we present the results of the evaluation, we first give details on the used data set.

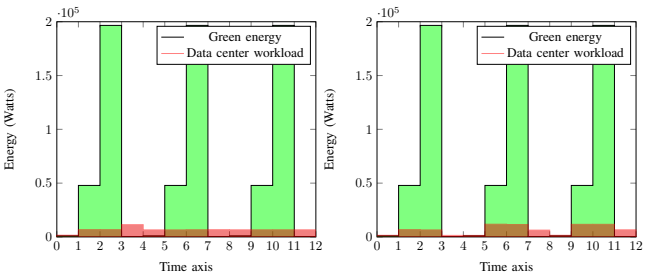
Data set description

Each of the 3 real workload data set is organized as follows:

(1) We have 55 servers, where each server is characterized by a limited amount of CPU and RAM resource.
(2) A set of 567 active applications to be encapsulated into virtual machines. As introduced in section III-D, each active application is executed throughout the time horizon. In our case, the time horizon is 12 hours. We see from the traces that the CPU and RAM needs of an active application may vary from one time slot to another. This translates into $6804 = 567 \times 12$ active tasks to handle. The migration cost in terms of energy consumption is also specified.
(3) A set of 2268 batch applications to be encapsulated into virtual machines. Each batch application has a fixed execution time, a slack value, a fixed start time and a migration cost. The requirements of the batch applications also vary from one time slot to another. This translates into more than 6500 batch tasks to handle.
(4) The renewable energy availability. Each data set includes a time series of length 12. This time series is the record of the renewable energy available every hour.

To evaluate our model and to follow the same protocol as in [15], we compared the results to those of a baseline first fit algorithm. The first fit algorithm tries to schedule the task as soon as possible when there is enough resources available. Both models were implemented in Prolog with the constraint solver Sicstus Prolog [10].

Figures 4a and 4b show how the model is able to move a considerable amount of the data center workload to slots where green energy is available.



(a) Baseline (b) GEASP

Fig. 4: Workload scheduling.

	Total E.C	Brown E.C.	Green E.C
Baseline	78811	39019	39792
GEASP Model	78694	18500	60194
	-0.14%	**-52.58%**	**+51.27%**

TABLE II: Energy saving on the first real trace (W).

	Total E.C	Brown E.C.	Green E.C
Baseline	78309	38699	39610
GEASP Model	78374	18196	60178
	+0.08%	**-51.98%**	**+51.92%**

TABLE III: Energy saving on the second real trace (W).

Tables II, III and IV give details on how the GEASP model significantly reduces the brown energy consumption and increases the green energy consumption.

	Total E.C	Brown E.C.	Green E.C
Baseline	79100	36195	42905
GEASP Model	78868	14670	64198
	-0.29%	**-59.46%**	**+49.62%**

TABLE IV: Energy saving on the third real trace (W).

On average compared to baseline, GEASP model reduces by 55% the brown energy consumption of the data center and increases by 50.9% its green energy consumption. Although PIKA [15] has a much higher green energy consumption as compared to the baseline algorithm, it does so with two major drawbacks. First, unlike PIKA [15] that uses opportunistic scheduling, our model finishes all the tasks not later than the baseline. This advantage comes from the time horizon that is fixed such that no task is allowed to be scheduled beyond it. If the time horizon constraint is dropped, then there will eventually be room for more optimization. For example, an amount of the workload at slot 11 of Figure 4b could be opportunistically postponed to a later slot where there is more green energy available. But this in turn will result to the tasks being finished later than with the baseline algorithm. Second, while the GEASP model slightly decreases the total energy consumption (Brown + Green), PIKA increases it by up to 31%. Most of this energy consumption increase occurs during slots that have a high green energy availability, therefore the green energy integration ratio is increased.

VI. Conclusion

Energy consumption and environmental issues have become major concerns over the recent years in cloud computing. In this paper, we introduced the green energy aware scheduling problem (GEASP) to optimize the energy consumption of a small/medium size data center. Using our model to solve the GEASP, we could optimize the energy consumption of a small/medium size data center in three ways. First, we slightly decrease its overall energy consumption, second we considerably decrease its brown energy consumption and finally we significantly increase its green energy consumption. To achieve these results, our solution relies mostly on batch applications whose execution can sometimes be delayed to periods with high green energy availability. Active applications on the other hand can not be suspended. Future work involves optimizing the energy consumption of active applications, by means of adapting the quality of service (QoS) with respect to the green energy availability [12].

Acknowledgments

This work has received a French state support granted to the CominLabs excellence laboratory and managed by the National Research Agency in the "Investing for the Future" program under reference Nb. ANR-10-LABX-07-01.

The authors would like to thank the EasyVirt company for providing them real workload traces from Cloud providers.

References

[1] E. Arafailova, N. Beldiceanu, R. Douence, M. Carlsson, P. Flener, M. A. F. Rodríguez, J. Pearson, and H. Simonis, "Global Constraint Catalog, Volume II, Time-Series Constraints," *CoRR*, 2016.

[2] P. Baptiste, C. Le Pape, and W. Nuijten, *Constraint-based scheduling: applying constraint programming to scheduling problems*. Springer Science & Business Media, 2012, vol. 39.

[3] L. A. Barroso, "The price of performance," *Queue*, 2005.

[4] N. Beldiceanu and M. Carlsson, "A new multi-resource cumulatives constraint with negative heights," in *CP*. Springer, 2002.

[5] N. Beldiceanu, B. D. Feris, P. Gravey, S. Hasan, C. Jard, T. Ledoux, Y. Li, D. Lime, G. Madi-Wamba, J.-M. Menaud *et al.*, "The epoc project: Energy proportional and opportunistic computing system," in *Int. Conf. on Smart Cities and Green ICT Systems (SMARTGREENS)*, 2015.

[6] ——, "Towards energy-proportional clouds partially powered by renewable energy," *Computing*, 2017.

[7] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and experience*, 2011.

[8] H. Cambazard, D. Mehta, B. O'Sullivan, and H. Simonis, "Bin packing with linear usage costs–an application to energy management in data centres," in *CP*. Springer, 2013.

[9] ——, "Constraint programming based large neighbourhood search for energy minimisation in data centres," in *International Conference on Grid Economics and Business Models*. Springer, 2013.

[10] M. Carlsson and al, *SICStus Prolog User´s Manual*, SICS Swedish ICT AB, Mai 2014.

[11] J. Hamilton, "Cooperative expendable micro-slice servers (cems): low cost, low power servers for internet-scale services," in *Conference on Innovative Data Systems Research (CIDR'09)*, 2009.

[12] M. S. Hasan, F. Alvares, T. Ledoux, and J.-L. Pazat, "Enabling green energy awareness in interactive cloud application," in *IEEE International Conference on Cloud Computing Technology and Science 2016*, 2016.

[13] F. Hermenier, S. Demassey, and X. Lorca, "Bin repacking scheduling in virtualized datacenters," in *International Conference on Principles and Practice of Constraint Programming*. Springer, 2011.

[14] A. Letort, N. Beldiceanu, and M. Carlsson, "A scalable sweep algorithm for the cumulative constraint," in *Principles and Practice of Constraint Programming*. Springer, 2012, pp. 439–454.

[15] Y. Li, A.-C. Orgerie, and J.-M. Menaud, "Opportunistic scheduling in clouds partially powered by green energy," in *IEEE International Conference on Green Computing and Communications*, 2015.

[16] A.-C. Orgerie, M. D. d. Assuncao, and L. Lefevre, "A survey on techniques for improving the energy efficiency of large-scale distributed systems," *ACM Computing Surveys (CSUR)*, 2014.

[17] F. Rossi, P. Van Beek, and T. Walsh, *Handbook of constraint programming*. Elsevier, 2006.

[18] P. Van Hentenryck and J.-P. Carillon, "Generality versus specificity: An experience with ai and or techniques." in *AAAI*, 1988, pp. 660–664.