A Carbon-aware Workload Dispatcher in Cloud Computing Systems

Tayebeh Bahreini
IBM T.J. Watson Research Center
Yorktown Heights, NY
tbahreini@ibm.com

Asser Tantawi
IBM T.J. Watson Research Center
Yorktown Heights, NY
tantawi@us.ibm.com

Alaa Youssef
IBM T.J. Watson Research Center
Yorktown Heights, NY
asyousse@us.ibm.com

***Abstract*—The amount of carbon emission associated with the computational energy consumption in data centers depends, in a significant way, on the schedule of the workloads. Due to the inconsistent availability of renewable energy over time, in addition to the existence of various sources of power in grid regions, the carbon intensity of data centers changes over time and location. Thus, the placement and scheduling of flexible workloads, based on the carbon intensity of power sources in data centers, can remarkably decrease the carbon emission. In this paper, we address the problem of placement and scheduling of workloads over geographically distributed data centers. We propose two algorithms that take the variability of carbon intensity of the power sources of the data centers, as well as their computational resource availability, into account when deciding about the placement and scheduling of the workloads. The first is a randomized rounding approximation algorithm that provides solutions that are guaranteed to be within a given distance from the optimal solution. The second is a sample-based algorithm that improves the solutions obtained by the randomized rounding approximation algorithm. The experimental results show that the proposed algorithms can solve the problem efficiently.**

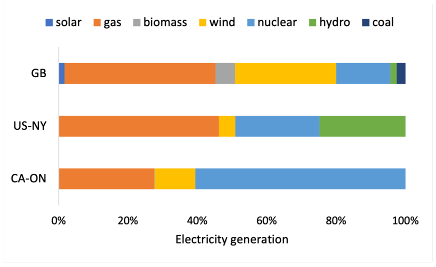
***Index Terms*—approximation algorithm, cloud sustainability, green computing, placement, randomized rounding, scheduling**

I. Introduction

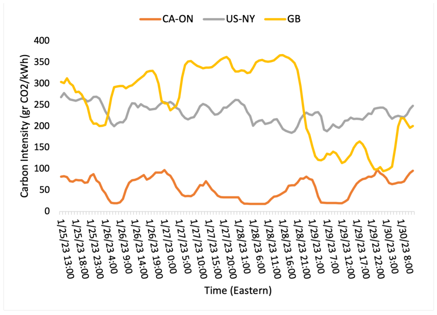
The electricity of data centers is supplied from various sources of renewable and non-renewable energy provided by the grid or on-site generators. Renewable sources have a low carbon emission, but there is uncertainty in their availability. To satisfy the electricity demand, the grid has to use both renewable and non-renewable sources. Figure 1a shows the average fraction of electricity generated by various sources of energy during January 2023 in three electricity grid zones: Markham in Ontario, Canada (CA-ON), Rochester in New York, US (US-NY), and Portsmouth in the UK (GB). We observe that the fraction of each renewable or non-renewable energy source varies across the regions. On the other hand, the availability of renewable energy in a region depends on weather conditions, thus is highly unstable. The electricity demand is also constantly changing and the grid may need to turn on/off generators to meet the demand. As a result, the carbon intensity, which is the weighted average of carbon emitted by each source, varies over time. Figure 1b shows how the hourly carbon intensity changes over five days in the three regions. Another challenge related to renewable energy is that, in the case of low demand, the power generated by renewable sources may be wasted as storing energy is expensive [10]. Thus, utilizing these sources of energy is of high importance. To reduce the carbon emission, cloud providers should consider this variation when they decide about the execution of their workloads. In other words, cloud providers can reduce carbon emission by properly placing and scheduling jobs through a choice of a data center and a particular time of day.

Several efforts have been devoted to minimizing the carbon footprint associated with the computational energy consumption in cloud systems. A body of research has considered job-level temporal shifting of workloads as an effective way of reducing carbon emission [1], [6], [7], [11]. The main idea of these approaches is to postpone the execution of some jobs until the carbon intensity (or other objectives such as cost and/or completion time) is low. For example, Liu et al. [11] studied shifting flexible workloads in order to take advantage of time variations of electricity price, the availability of renewable energy, and the efficiency of cooling. They considered several hours to multiple days as deadline of the jobs. In contrast to temporal shifting, many studies addressed the spatial workload placement among geographically distributed data centers [2], [5], [15]. The goal of these approaches is to execute workloads at data centers where the objective value (e.g., carbon footprint, QoS, or cost) is optimized. For example, Doyle et al. [5] developed a distributed algorithm to decide about the target server for workloads with the objective of minimizing carbon footprint of executing the workloads while QoS is satisfied. There are also few treatments in the literature that considered both time and space dimensions when they decide about the schedule of jobs [13]. In addition to job-level optimization, some studies have focused on load-level optimization [4], [12], in which, based on the hourly demand as well as the carbon intensity/availability of renewable energy in each hour, the optimizer adjusts the capacity. For example, Google's carbon intelligent computing system [12] reduces the available capacity at particular hours of the day when the grid carbon intensity is high or it brings a high cost to the system.

In this paper we consider both job-level and load-level optimization by addressing the problem of Carbon-aware Job Placement and Scheduling (CJPS). We consider both temporal and spatial scheduling of the workloads while taking into account anticipated future load. Our aim is to maximize



(a) Average electricity production by source.



(b) Hourly carbon intensity.

Fig. 1: Power and carbon data in three grid regions.

the usage of computing resources when and where the carbon intensity of power sources of data centers is low. We consider a time slotted system, in which information about queued jobs in the current time slot is available. Although there is a high uncertainty in predicting the characteristics of each individual job that may arrive in the future, the total resource demand is predictable. Thus, in this paper, we consider the forecasted resource demand while we decide about the current jobs in the queue. We develop a randomized rounding approximation algorithm (RR-APX) to solve CJPS. Our approximation algorithm guarantees that it obtains a solution with the expected objective value (total green resource utilization) not less than the optimal value, while the probability of violating the constraints on the resources is bounded by a small factor. We also develop a sample-based rounding algorithm (SRR-APX) to improve the solution obtained by RR-APX.

II. Carbon-aware Job Placement and Scheduling

We address the CJPS problem for a cloud system with multiple geographically distributed data centers. We consider a time slotted system in which a set of jobs $\mathcal{N}$ is to be scheduled on a set of data centers $\mathcal{M}$ within a look-ahead time horizon T, with the aim of maximizing the usage of computing resources when and where low-carbon energy is available. The notation we use in the formulation is provided in Table I.

Each job $i \in \mathcal{N}$ is characterized by the tuple (r_i, l_i, d_i), where r_i is the required amount of the computational resource, l_i is the run time, and d_i is the deadline. All parameters r_i, l_i, and d_i are integers. Here, we consider a single type of computational resource, namely CPU, but the formulation can be extended to multiple types. We assume that the preemption of jobs is not allowed. Each data center $j \in \mathcal{M}$ is characterized by $(\{B_j^t, I_j^t\})$, where B_j^t is the available CPU resources and I_j^t is the carbon intensity in time slot t. We assume that the length of time slots is fixed and the carbon intensity and available resources do not change within a time slot.

Due to the limited capacity, the decision maker may not be able to allocate all jobs, rather it tries to allocate as many jobs as possible (see the objective function). We assume that, after a job starts in a data center, it is not allowed to migrate to another data center. We define a binary variable x_{ij}^t which is one if job i starts at time slot t in data center j, and zero otherwise. Thus, we have

$$\sum_{j\in\mathcal{M}}\sum_{t\in\mathcal{T}} x_{ij}^t \leq 1 \quad \forall i \in \mathcal{N}, \tag{1}$$

$$x_{ij}^t = 0 \quad \forall i \in \mathcal{N}, j \in \mathcal{M}, t \in \mathcal{T}, \text{and } t \geq d_i - l_i. \tag{2}$$

For future load, we consider more relaxed scheduling/placement constraints. We denote by $R^{t'}$ the total CPU demand in time slot t', and by $z_j^{tt'}$, a continuous variable, the fraction of demand of time slot t' that is allocated in time slot t ($t \geq t'$). The total demand of time slot t' can be allocated over all data centers and time slots. Thus, we have

$$\sum_{j\in\mathcal{M}}\sum_{t\in\mathcal{T}} z_j^{tt'} \leq 1 \quad \forall t' \in \mathcal{T}, \tag{3}$$

$$z_j^{tt'} = 0 \quad \forall j \in \mathcal{M},\ t, t' \in \mathcal{T}, \text{and } t' > t. \tag{4}$$

For each data center j, the total amount of resources allocated for the jobs plus the total resources reserved for future load cannot exceed its available capacity, i.e.,

$$\sum_{i\in\mathcal{N}}\sum_{t'\in\tau_i^t} x_{ij}^{t'} \cdot r_i + \sum_{t'\in\mathcal{T}} z_j^{tt'} \cdot R^{t'} \leq B_j^t \ \forall j \in \mathcal{M}, t \in \mathcal{T}. \tag{5}$$

In the above formulation, the total resource allocation of the jobs in time slot t on data center j is obtained based on the resource requirement of the jobs running on that data center in that particular time slot. Job i is running on data center j in time slot t if it has been started in any time slots in $\tau_i^t = \{t - l_i + 1, \ldots, t\}$.

The goal of CJPS is to maximize the CPU utilization when and where the carbon intensity is low. For this purpose, we define Green Resource Utilization (GRU) as a ratio between the CPU utilization over the carbon intensity,

$$GRU_j^t = \frac{\sum_{i\in\mathcal{N}} \sum_{t'\in\tau_i^t} x_{ij}^{t'} \cdot r_i + \gamma \cdot \sum_{t'\in\mathcal{T}} z_j^{tt'} \cdot R^{t'}}{I_j^t}. \tag{6}$$

Here, γ is a prioritization factor, where $\gamma = 1$ means that we consider the same priority of using green resources for jobs in the queue and for the future load. And, $\gamma < 1$ means that jobs in the queue have a higher priority of using low-carbon resources. The objective of CJPS is to maximize the

TABLE I: Notation

Notation	Description
T	Time horizon for executing the jobs.
$\mathcal{M}$	Set of m data centers.
$\mathcal{N}$	Set of n jobs currently in the queue.
B_j^t	Available capacity of data center j in time slot t.
I_j^t	Carbon intensity of data center j in time slot t.
r_i	Computational resource requested by job i.
R^t	Total CPU demand in time slot t.
l_i	Duration time of job i.
d_i	Deadline to complete job i.
$\mathcal{T}$	Set of time slots in the look-ahead time horizon T.
x_{ij}^t	Binary variable, it is 1 if job i is started on data center j in time slot t; and 0, otherwise.
$z_j^{tt'}$	Continuous variable indicating the fraction of load of time slot t' allocated in time slot t on data center j.
γ	Prioritization factor.

total value of Green Recourse Utilization (GRU) over all data centers and over all time slots. It is given by,

$$\max \sum_{j\in\mathcal{M}}\sum_{t\in\mathcal{T}} \frac{\sum_{i\in\mathcal{N}} \sum\limits_{t'\in\tau_i^t} x_{ij}^{t'}\cdot r_i + \gamma\cdot \sum\limits_{t'\in\mathcal{T}} z_j^{tt'}\cdot R^{t'}}{I_j^t}.$$

Complexity. We prove that CJPS is NP-hard by showing a special case of the problem is NP-hard. We consider a special case of CJPS, namely S-CJPS, with a single data center and a fixed carbon intensity I and capacity B over all time slots. We assume that the deadline of all jobs is T, i.e., $d_i = T$, and there is no future load. For this special case, the optimal solution is the solution that maximizes the total usage of CPU over all time slots. Let us represent the data center with a large rectangle of width T and height B. We also represent each job i with a rectangle of width l_i and height r_i. Finding an optimal solution for S-CJPS is equivalent to packing a subset of rectangles $\{(l_i, r_i)\}$ into a larger rectangle with width T and height B so that the maximum space of the large rectangle is used. This problem belongs to the class of rectangle packing problems [8] that are known as NP-hard problems. This implies that finding the optimal solution for S-CJPS is NP-hard. Thus, CJPS in general is NP-hard.

III. Proposed algorithms for job placement and scheduling problem

Given that CJPS is NP-hard, the integer program of the problem (CJPS-MILP) cannot be solved in polynomial time unless $P = NP$. However, Linear Programs (LPs) are polynomial-time solvable. In this section, we introduce two LP-based algorithms for solving CJPS. The first algorithm is a randomized rounding approximation algorithm, called RR-APX. It is based on the LP relaxation of CJPS-MILP. We show that the expected value of the objective function obtained by RR-APX is not less than the objective value of the optimal solution. However in each time slot, the total allocation may exceed the available resources with some probability. To deal with the capacity violation constraints, we develop SRR-APX, a sample-based algorithm that has RR-APX in its core and tries to improve the solution obtained by RR-APX using sampling technique.

Algorithm 1 RR-APX Algorithm

```
Input: $(\mathcal{N}, M)$
1: $X \leftarrow \{0\}$
2: $(\bar{X}, Z) \leftarrow$ LP-Solver$(\mathcal{N}, \mathcal{M}\})$
3: for each $i \in \mathcal{N}$ do
4:     $(j^*, t^*) \leftarrow$roll-dice$(i, \bar{X})$
5:     $x_{ij^*}^{t^*} \leftarrow 1$
6: $GRU \leftarrow \sum_{j\in\mathcal{M}}\sum_{t\in\mathcal{T}} \frac{\sum_{i\in\mathcal{N}} \sum_{t'\in\tau_i^t} x_{ij}^{t'}\cdot r_i+\gamma\cdot \sum_{t'\in\mathcal{T}} z_j^{tt'}\cdot R^{t'}}{I_j^t}$
7: $X \leftarrow X_{best}$
Output: $(GRU, X)$
```

A. RR-APX Algorithm

RR-APX is built based on the LP relaxation of CJPS-MILP in which constraints (2) are relaxed to,

$$0 \le x_{ij}^t \le 1, \quad \forall i\in\mathcal{N}, j\in\mathcal{M}, t\in\mathcal{T}, \text{and } t\ge d_i - l_i. \quad (7)$$

Let us denote by $(\bar{X}, Z) = (\{\bar{x}_{ij}^t\}, \{z_j^{tt'}\})$, the optimal solution to the LP relaxation. We do not touch the resource reservation for future load obtained by the LP solution, i.e., Z, but we would like to round fractional values of $\{\bar{x}_{ij}^t\}$ to either 0 or 1. Our goal is to find an integer solution $X = \{x_{ij}^t\}$, without increasing the gap to the optimal solution too much. An algorithmic description of RR-APX is given in Algorithm 1. The input to RR-APX is the information of the set of jobs and the set of data centers. The output consists of the scheduling matrix and GRU. First, the algorithm initializes the scheduling matrix with zero (Line 1). Then, it obtains the fractional solution $(\bar{X}, Z)$ by solving the LP relaxation of CJPS-MILP (Line 2). Independently, for each job i, the algorithm calls roll-dice procedure. The roll-dice procedure rolls a dice that has $M \times T$ sides of $\{(j,t)\}$, in which the probability of getting side (j,t) is $\bar{x}_{ij}^t$. The output of the procedure is the outcome of the rolling. It sets $x_{ij^*}^{t^*} = 1$, if the outcome of the rolling is (j^*, t^*) (Lines 3-5). Then, RR-APX calculates the value of GRU (Line 6) and returns it along with the scheduling matrix.

1) Properties: First, we discuss the time complexity of RR-APX and show that the algorithm is polynomial. Then, we provide approximation bounds for the algorithm. For a given input, RR-APX obtains an integer solution with the GRU not less than the optimal value. However, the solution may violate the capacity constraints. The probability of having total allocation greater than or equal $(1+\epsilon)\cdot B_j^t$ on data center j in time slot t is bounded by $\frac{1}{(1+\epsilon)}$.

Time complexity. The most time consuming part of the algorithm is solving the LP relaxation of CJPS-MILP, which takes polynomial time [9]. The other parts of the algorithm (Lines 5-6) also have polynomial time complexity. Therefore, the time complexity of the algorithm is polynomial.

Lemma 1. *The expected value of GRU obtained by RR-APX is not less than the optimal value OPT.*

Proof. Let us denote by $E[GRU(X,Z)]$ the expected value of the total green resource utilization obtained by RR-APX and by P the probability of the happening of an event,

$$E[GRU(X,Z)] = \sum_{j\in\mathcal{M}}\sum_{t\in\mathcal{T}} \frac{\sum_{i\in\mathcal{N}} \sum\limits_{t'\in\tau_i^t} P(x_{ij}^{t'}=1)\cdot r_i}{I_j^t} + \sum_{j\in\mathcal{M}}\sum_{t\in\mathcal{T}} \frac{\gamma\cdot\sum\limits_{t'\in\mathcal{T}} z_j^{tt'}\cdot R^{t'}}{I_j^t} \geq \sum_{j\in\mathcal{M}}\sum_{t\in\mathcal{T}} \frac{\sum_{i\in\mathcal{N}} \sum\limits_{t'\in\tau_i^t} \bar{x}_{ij}^{t'}\cdot r_i}{I_j^t} + \sum_{j\in\mathcal{M}}\sum_{t\in\mathcal{T}} \frac{\gamma\cdot\sum\limits_{t'\in\mathcal{T}} z_j^{tt'}\cdot R^{t'}}{I_j^t} \geq LP^* \geq OPT,$$

where LP^* is the objective value of the LP relaxation. □

Now, we introduce Markov's inequality to be able to analyze the feasibility of the solutions obtained by RR-APX. In simple words, Markov's inequality says that for a non-negative random variable Y and any positive real number a, the probability that Y is at least a is less than or equal to the expected value of Y divided by a.

Lemma 2 (Markov's inequality)**.** *Let Y be a random variable that takes only non-negative values and $a>0$, then,*

$$P[Y\geq a] \leq \frac{E[Y]}{a}. \tag{8}$$

Lemma 3. *The probability that the total resource allocation obtained by RR-APX in time slot t on data center j is at least $(1+\epsilon)\cdot B_j^t$ is bounded by $\frac{1}{(1+\epsilon)}$, where $\epsilon>0$. i.e.,*

$$P\left[\sum_{i\in\mathcal{N}}\sum_{t'\in\tau_j^t} r_i\cdot x_{ij}^{t'} + \sum_{t'\in\mathcal{T}} z_j^{tt'}\cdot R^{t'} \geq (1+\epsilon)\cdot B_j^t\right] \leq \frac{1}{(1+\epsilon)} \quad \forall j\in\mathcal{M}, t\in\mathcal{T}. \tag{9}$$

Proof. We define a random variable A_j^t as the total allocation on data center j in time slot t, i.e.,

$$A_j^t = \sum_{i\in\mathcal{N}}\sum_{t'\in\tau_i^t} r_i\cdot x_{ij}^{t'} + \sum_{t'\in\mathcal{T}} z_j^{tt'}\cdot R^{t'}. \tag{10}$$

The expected value of A_j^t is obtained as follows,

$$E[A_j^t] = \sum_{i\in\mathcal{N}}\sum_{t'\in\tau_j^t} r_i\cdot P[x_{ij}^{t'}=1] + \sum_{t'\in\mathcal{T}} z_j^{tt'}\cdot R^{t'} \leq \sum_{i\in\mathcal{N}}\sum_{t'\in\tau_j^t} r_i\cdot \bar{x}_{ij}^{t'} + \sum_{t'\in\mathcal{T}} z_j^{tt'}\cdot R^{t'} \leq B_j^t. \tag{11}$$

The last inequality comes from the fact that $(\bar{X},Z)$ is a feasible solution for the LP relaxation of CJPS-MILP. On the other hand, A_j^t takes on non-negative values. Thus, according to Markov's inequality,

$$E[A_j^t] \geq \big((1+\epsilon)\cdot B_j^t\big)\cdot P[A_j^t \geq (1+\epsilon)\cdot B_j^t]. \tag{12}$$

Therefore, we can conclude that,

$$P[A_j^t \geq (1+\epsilon)\cdot B_j^t] \leq \frac{E[A_j^t]}{(1+\epsilon)\cdot B_j^t} \leq \frac{1}{(1+\epsilon)}. \tag{13}$$

□

Based on Lemma 1 and Lemma 3, we can conclude the following theorem.

Theorem 1. *The RR-APX algorithm obtains an integer solution for CJPS-MILP with the expected value of green resource utilization not less than optimal value while the probability of having total allocation greater than or equal $(1+\epsilon)\cdot B_j^t$ in any time slot and data center is bounded by $\frac{1}{(1+\epsilon)}$.*

B. *SRR-APX Algorithm*

In Subsection III-A1, we showed that the solution obtained by RR-APX may violate the capacity constraints with some probability. To improve the solution, we use a sampling technique in which, instead of getting one sample from the fractional solution $\bar{X}$, we get S samples. After obtaining a fractional solution, the algorithm rolls the dice for each job independently and repeats this procedure for S times to reduce the risk of over allocation and obtain a better solution. Here, by better solution, we mean the solution in which a lower rate of job failures occurs. In RR-APX, we guarantee that all jobs are scheduled and finished before their deadline, while there is a probability of over allocation. Different from RR-APX, SRR-APX seeks to find a solution that guarantees that no over allocation happens; but some jobs may fail to be scheduled. From a practical point of view, SRR-APX is more favorable when there is no auto-scaling option in the cluster.

SRR-APX is given in Algorithm 2. The algorithm first initializes the variables (Lines 1-3). Here, F_{best} saves the number of failed jobs in the best solution, GRU_{best} saves the associated GRU value, and X_{best} saves the scheduling matrix. The algorithm obtains the fractional solution for CJPS-MILP by calling the LP-solver procedure (Line 4). Then, it generates S random solutions from the fractional solution $\bar{X}$ through a loop (Lines 5-25). Variable F saves the number of failed jobs of the current solution, X is the scheduling matrix, and A is a matrix saving the total allocation on each data center in each time slot. Each solution is built by calling roll-dice procedure, independently for each job (Line 10). If by scheduling the current job i, the total allocation exceeds the capacity, the job is dropped and accordingly, the number of failed jobs is updated (Lines 11-17). If violation does not happen, the algorithm updates the scheduling matrix and the allocation matrix (Lines 18-21). After rolling the dice for all jobs, the algorithm compares the solution obtained by the current sample with the best solution so far. The one that has the minimum number of failures is the desirable solution. If two samples have the same number of failures, the algorithm picks the one that has the highest GRU value (Lines 22-25). Finally, the algorithm returns the value of GRU of the best solution along with the scheduling matrix X.

Algorithm 2 SRR-APX Algorithm

```
Input: $(\mathcal{N}, M)$
1: $F_{best} \leftarrow N$
2: $GRU_{best} \leftarrow 0$
3: $X_{best} \leftarrow \{0\}$
4: $(\bar{X}, Z) \leftarrow$ LP-Solver$(\mathcal{N}, \mathcal{M})$
5: for each $s \in \mathcal{S}$ do
6:     $F \leftarrow 0$
7:     $X \leftarrow \{0\}$
8:     $A \leftarrow \{0\}$
9:     for each $i \in \mathcal{N}$ do
10:        $(j^*, t^*) \leftarrow$ roll-dice$(i, \bar{X})$
11:        violated$\leftarrow$ False
12:        for $t \in \{t^*, \dots t^* + l_i - 1\}$ do
13:            if $A_{j^*}^{t^*} + r_i > B_{j^*}^t$ then
14:                violated$\leftarrow$ True
15:                break
16:        if violated then
17:            $F \leftarrow F + 1$
18:        else
19:            $x_{ij^*}^{t^*} \leftarrow 1$
20:            for $t \in \{t^*, \dots, t^* + l_i - 1\}$ do
21:                $A_{j^*}^t \leftarrow A_{j^*}^t + r_i$
22:     if $F < F_{best}$ or $(F = F_{best}$ and $GRU(X, Z) > GRU_{best})$ then
23:        $F_{best} \leftarrow F$
24:        $GRU_{best} \leftarrow GRU(X, Z)$
25:        $X_{best} \leftarrow X$
26: $GRU \leftarrow GRU_{best}$
27: $X \leftarrow X_{best}$
Output: $(GRU, X)$
```

IV. Experimental Analysis

For our experiments, we use a two month workload trace collected from a production cluster in Alibaba PAI [14]. The workloads are a mix of training and inference jobs. From the trace, we choose long-running time jobs with run time between 30 minutes to five hours. We exclude short-running jobs from the experiments as they usually require a short deadline and cannot tolerate long delays. Furthermore, the carbon intensity of clusters does not change quickly. Thus, the impact of scheduling short-running jobs may not be significant. We set the length of each time slot to 30 minutes and extract the information of the jobs from randomly 70 consecutive time slots (35 hours) from the trace to simulate the arrival of the jobs as well as future load. We use publicly available web services that provide carbon intensity data and consider three electricity grid zones: Markham in Ontario, Canada, Rochester in New York, US, and Portsmouth in the UK.

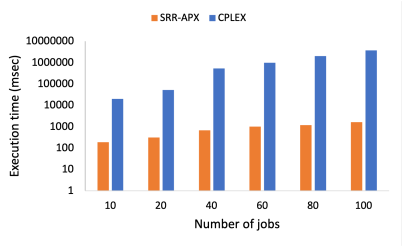
Our experimental results include two parts. The first part of the experiments is for small size problem instances. The aim of this part is to compare the solutions obtained by SRR-APX with the optimal solution obtained by CPLEX solver, which is not able to solve large-scale problems in a reasonable amount of time. To stress test both SRR-APX and CPLEX, we generate random jobs (to provide more heterogeneity) in which the run time as well as the resource requirement of jobs is chosen from a random uniform distribution $U[1\ 10]$. In the second part, we consider more realistic environments, in which the scheduler runs periodically over time slots. For this part, we generate jobs using the Alibaba traces. In this set of experiments, the scheduler is run over 50 time slots. To evaluate the scalability of SRR-APX, we fix the available capacity and vary the number of submitted jobs from 1,000 to 6,000 jobs, in order to simulate various load in the system. For this size of problem, CPLEX is not able to be run in a reasonable amount of time. Thus, we do not compare SRR-APX with CPLEX. We consider a variant of SRR-APX in which reserving resources for the future load is not considered when the optimizer decides about the schedule of the current jobs. We do this by setting $\gamma = 0$. We consider this variant as a baseline and compare its solutions to the solutions obtained by SRR-APX algorithm for $\gamma = 1$, in which jobs are scheduled with reserving resources for future load.

SRR-APX and CPLEX are implemented in python version 3.6.0 and executed on an Intel 2.3 GHz Core i9 with 64 GB RAM system. For the experiments involving CPLEX we used the CPLEX 12.8 solver provided by IBM ILOG CPLEX [3]. Each experiment is performed ten times and the analysis is performed based on the average value of the metrics.

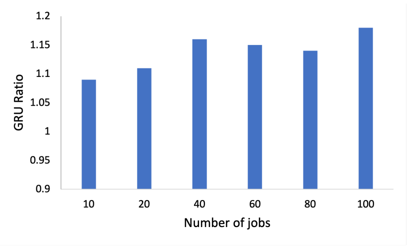
First we compare the quality of solutions obtained by SRR-APX to that of the solutions obtained by using the CPLEX solver, to solve CJPS-MILP. We fix the capacity to 50 cores, i.e., $B_j^t = 50$, and vary the load by increasing the number of jobs submitted to the system between $n = 10$ and $n = 100$. We chose this type of instances in order to be able to solve them optimally using CPLEX and compare the performance of our algorithm with that of the optimal solution.

Figure 2a shows the average execution time of CPLEX and that of SRR-APX on a logarithmic scale. For each instance, the average execution time of CPLEX is several orders of magnitude higher than the execution times of SRR-APX. For example, in the case of instances with $n = 10$, the average execution time of SRR-APX is around 186 milliseconds, while for CPLEX it is around 12,181 milliseconds. We also observe an increase in the execution time of both approaches with the increase in the number of jobs. SRR-APX shows a polynomial growth of the running time with the increase of the number of jobs, while CPLEX shows an exponential behaviour. The reason is that the time complexity of the SRR-APX algorithm is polynomial, while the time complexity of CPLEX to find the exact solution is exponential.

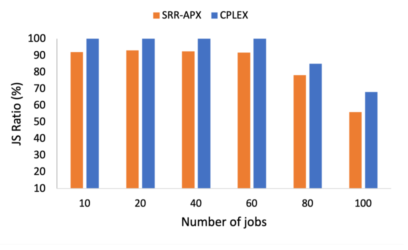
We define a performance metric called $GRU\ ratio = \frac{GRU^*}{GRU}$ as the ratio between the objective value obtained by CPLEX, GRU^*, and that obtained by SRR-APX, GRU. We also define Job Satisfaction ratio $JS\ ratio$ as the percentage of jobs that are executed before their deadline. Figure 2b shows the average $GRU\ ratio$ for each problem instance. We observe that this ratio varies between 1.09 and 1.2 which is an acceptable range. We do not observe any significant increase



(a) Execution time.

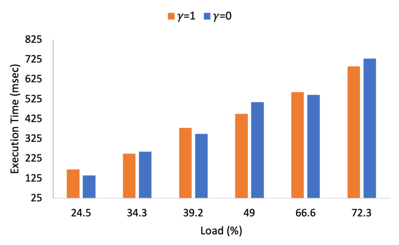


(b) Green resource utilization ratio.

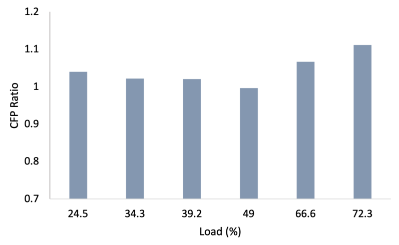


(c) Job satisfaction ratio.

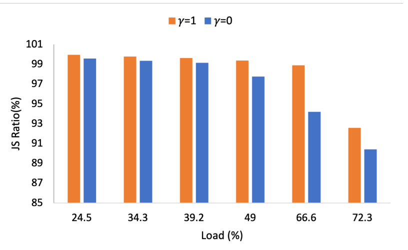
Fig. 2: SRR-APX vs. CPLEX.



(a) Execution time.



(b) Carbon footprint ratio.



(c) Job satisfaction ratio.

Fig. 3: Impact of resource reservation on performance of SRR-APX.

in the $GRU\ ratio$ with the increase in the number of jobs. Figure 2c shows the average $JS\ ratio$ obtained by CPLEX and SRR-APX for each problem instance. We observe that $JS\ ratio$ obtained by SRR-APX is very close to that of CPLEX for all sizes of instances. With the increase of the load, $JS\ ratio$ of both approaches decreases as lower resources are available per job.

Next, we investigate the scalability and performance of the SRR-APX algorithm for large size problem instances. We fix the available capacity of each cluster to 700 cores and run the algorithm over 50 time slots. We vary the number of jobs submitted over 50 time slots between 1,000 to 6,000 to have load between 25% to 70%. For this experiment, we consider two classes of problem instances. In the first class, we set $\gamma = 1$ to give the same weight to the current job and future load when deciding on the scheduling and the placement. In the second class, we set $\gamma = 0$, forcing the optimizer to neglect the future load. We investigate the impact of the value of γ on the scheduling and placement of the jobs.

Figure 3a shows the execution time of the algorithm for both cases. We observe that the execution time of the algorithm polynomially increases with the increase of load (number of jobs). For all cases the average execution time of the algorithm is less than one second that makes the algorithm suitable for online scheduling in cloud environment. We define Carbon FootPrint ratio, $CFP\ ratio$, as the ratio of the total carbon footprint of executing jobs obtained by SRR-APX with $\gamma = 1$ over that obtained by SRR-APX with $\gamma = 0$. Figure 3b shows the average $CFP\ ratio$ per time slot. Here, we observe that for load less than 66.6%, $CFP\ ratio$ decreases with the increase of the load. This indicates that by the increase of the load, the carbon footprint obtained with considering future load is lower than that without considering future load. Then, for load > 0.66, we observe that with the increase of the load, this ratio increases as well. The reason is that for these problem instances, a relatively higher percentage of the jobs fail when future load is not considered, as illustrated in Figure 3c. Thus, fewer jobs are executed and lower carbon footprint is obtained for this class yield in a higher $CFP\ ratio$. Figure 3c shows the average $JS\ ratio$ for various load. We observe that the job satisfaction ratio decreases with the increase of the load. The reason is that with the increase of the load, a lower rate of the jobs can be allocated due to the limited capacity and we expect a higher rate of job failures. We also observe that for each problem instance, with the case of $\gamma = 1$, a higher job satisfaction ratio is obtained compared to the case with $\gamma = 0$.

V. Conclusion and Future Work

In this paper, we addressed the problem of carbon-aware placement and scheduling of workloads over geographically distributed data centers. We proved that the problem is NP-hard and developed two efficient algorithms, RR-APX and SRR-APX, for solving the problem. The RR-APX algorithm is an LP-based approximation algorithm that guarantees strong approximation bounds on the constraints and the objective function. The SRR-APX algorithm is a sample-based algorithm that has RR-APX in its core. We performed extensive experiments, based on real data traces and simulations, to investigate the performance of the SRR-APX algorithm. The results of these experiments indicated that the SRR-APX algorithm obtains high quality solutions and results in very low execution times. For future work, we plan to design a green scheduler for Kubernetes environment and deploy the SRR-APX algorithm to dispatch workloads over multi clusters based on the availability of low-carbon energy.

References

[1] T. Bahreini, A. Tantawi, and A. Youssef. An approximation algorithm for minimizing the cloud carbon footprint through workload scheduling. In *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*, pages 522–531, 2022.

[2] C. Chen, B. He, and X. Tang. Green-aware workload scheduling in geographically distributed data centers. In *4th IEEE international conference on cloud computing technology and science proceedings*, pages 82–89. IEEE, 2012.

[3] I. I. Cplex. V12. 8: User's manual for cplex. *International Business Machines Corporation*, 46(53):157, 2009.

[4] X. Deng, D. Wu, J. Shen, and J. He. Eco-aware online power management and load scheduling for green cloud datacenters. *IEEE Systems Journal*, 10(1):78–87, 2014.

[5] J. Doyle, D. O'Mahony, and R. Shorten. Server selection for carbon emission control. In *Proceedings of the 2nd ACM SIGCOMM workshop on Green networking*, pages 1–6, 2011.

[6] Í. Goiri, K. Le, M. E. Haque, R. Beauchea, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini. Greenslot: scheduling energy consumption in green datacenters. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11, 2011.

[7] Í. Goiri, K. Le, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini. Greenhadoop: leveraging green energy in data-processing frameworks. In *Proceedings of the 7th ACM european conference on Computer Systems*, pages 57–70, 2012.

[8] K. Jansen and G. Zhang. Maximizing the total profit of rectangles packed into a rectangle. *Algorithmica*, 47:323–342, 2007.

[9] L. G. Khachiyan. A polynomial algorithm in linear programming. In *Doklady Akademii Nauk*, volume 244, pages 1093–1096. Russian Academy of Sciences, 1979.

[10] W. Li, T. Yang, F. C. Delicato, P. F. Pires, Z. Tari, S. U. Khan, and A. Y. Zomaya. On enabling sustainable edge computing with renewable energy resources. *IEEE Communications Magazine*, 56(5):94–101, 2018.

[11] Z. Liu, Y. Chen, C. Bash, A. Wierman, D. Gmach, Z. Wang, M. Marwah, and C. Hyser. Renewable and cooling aware workload management for sustainable data centers. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE joint international conference on Measurement and Modeling of Computer Systems*, pages 175–186, 2012.

[12] A. Radovanovic, R. Koningstein, I. Schneider, B. Chen, A. Duarte, B. Roy, D. Xiao, M. Haridasan, P. Hung, N. Care, et al. Carbon-aware computing for datacenters. *arXiv preprint arXiv:2106.11750*, 2021.

[13] I. Rocha, C. Göttel, P. Felber, M. Pasin, R. Rouvoy, and V. Schiavoni. Heats: Heterogeneity-and energy-aware task-based scheduling. In *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pages 400–405. IEEE, 2019.

[14] Q. Weng, W. Xiao, Y. Yu, W. Wang, C. Wang, J. He, Y. Li, L. Zhang, W. Lin, and Y. Ding. MLaaS in the wild: Workload analysis and scheduling in large-scale heterogeneous GPU clusters. In *19th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 22)*, 2022.

[15] Y. Zhang, Y. Wang, X. Wang, et al. Greenware: Greening cloud-scale data centers to maximize the use of renewable energy. In *Middleware*, volume 7049, pages 143–164. Springer, 2011.