An Approximation Algorithm for Minimizing the Cloud Carbon Footprint through Workload Scheduling

1st Tayebeh Bahreini
IBM T.J. Watson Research Center
Yorktown Heights, NY
tbahreini@ibm.com

2nd Asser Tantawi
IBM T.J. Watson Research Center
Yorktown Heights, NY
tantawi@us.ibm.com

3rd Alaa Youssef
IBM T.J. Watson Research Center
Yorktown Heights, NY
asyousse@us.ibm.com

***Abstract*—In this paper, we address the problem of workload scheduling in data centers, while considering the greenness of the power sources. We prove that finding a feasible solution for the problem is NP-hard. Therefore, we develop an LP-based approximation algorithm to solve the problem in polynomial time. The proposed algorithm provides strong approximation bounds on the constraints and the objective of the problem. We conduct an extensive experimental analysis to evaluate the performance of the proposed algorithm using real world data.**

***Index Terms*—cloud optimization, cloud sustainability, green computing, approximation algorithm, resource constrained scheduling**

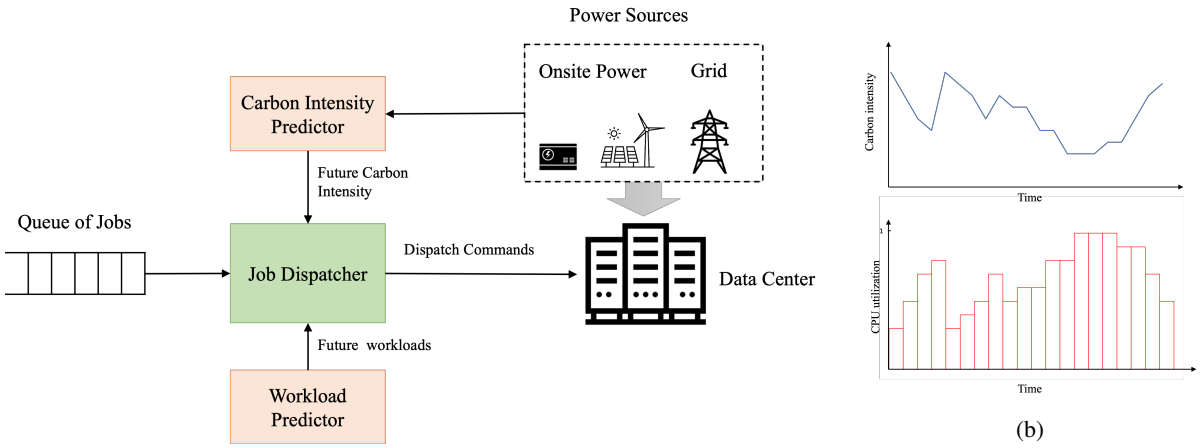
I. Introduction

Cloud computing is one of the most promising paradigms that provides scalable computing services to users. The core advantage of cloud computing stems from the powerful infrastructure that includes large data centers hosting several thousands of servers. In the last decade, with the dramatically increasing demands for the cloud services, the energy consumption of data centers has continued to accelerate rapidly. Data centers consume around 1% of the global electricity [19], contributing to 0.3% of the global CO2 emissions [4]. Thus, to improve the environmental sustainability, cloud providers must manage energy consumption of data centers and utilize more renewable energy to reduce carbon emissions. This is a multi-pronged endeavor, ranging from facility and cooling improvements, use of more efficient hardware technologies, to workload management. In this paper we are concerned with the latter.

Cloud providers such as Amazon, Google, and IBM have multiple geographically distributed data centers to provide highly reliable services and more transparency to users. Having data centers across various regions allows cloud providers to better utilize renewable energy. In fact, due to the variability in renewable energy across regions and over time, the greenness of the power sources of data centers varies by location and time. This is an opportunity for cloud providers to exploit this variation of greenness and allocate resources to the users when and where the amount of available renewable energy is relatively high. Fortunately, real time data about the power source mix to data centers is becoming increasingly available [4], and future predictions are also viable [20], [23]. This provides an opportunity to minimize the carbon footprint by properly placing and scheduling workloads through a choice of a data center and a particular time of day.

Due to the limited capacity of computing resources, it may not be always feasible to serve all the requests while guaranteeing their Quality of Service (QoS) requirements. In general, placement and scheduling of workloads, when limited computational capacity is available, is considered as an intractable problem. It can be shown that, finding a feasible solution for this problem is NP-hard, unless $P = NP$. The problem becomes even more complicated when, in addition to finding a feasible solution, an objective function (e.g, total carbon footprint generated by executing workloads) has to be optimized.

In this paper, we address the problem of resource constrained workload scheduling in cloud systems. We consider a cloud computing system that is subjected to a batch job workload. We design a dispatcher which schedules jobs over a time horizon, in such a way that the total carbon footprint generated by executing the jobs is minimized. Figure 1a depicts a high level architecture of the system. The dispatcher gets the future predictions of carbon intensity of the data centers, the future predictions of workloads, and accordingly decides about the order of execution of the jobs that are currently in the queue. The objective of the dispatcher is to minimize the overall carbon footprint related to processing the workload, while satisfying the constraints, namely job deadlines and computational capacity of data centers. We use the terms workload and job interchangeably throughout the paper. Figure 1b shows the output expected from the job dispatcher. With the increase in carbon intensity, the CPU utilization decreases, and with the decrease in carbon intensity, the CPU utilization increases. However, in practice, since jobs have different arrival times and deadlines and they have heterogeneous resource requirements, the CPU utilization may not be the same as in the ideal conditions, in which a fraction of a job can be executed at any time.



(a) (b)

Fig. 1: (a) System architecture of job execution. (b) CPU utilization under ideal conditions.

A. *Our contribution*

We address the workload scheduling problem in cloud systems, where the objective is to minimize the total carbon footprint related to processing of the workload. We show that finding a feasible solution for the problem is NP-hard. Thus, we develop an LP-based algorithm to solve the problem in polynomial time. The algorithm guarantees that in case of feasibility, it finds a schedule with the total carbon footprint not greater than the optimal value, while the violation of the constraints on the resources and the execution of the workloads is bounded by a small factor. The proposed approach provides fast solutions and makes it suitable for execution in real-time settings. We perform an extensive experimental analysis to show the performance of the proposed algorithm.

B. *Organization*

The rest of the paper is organized as follows. In Section II, we review the related work. In Section III, we define the optimization problem and discuss its complexity. In Section IV, we present the proposed algorithm for solving the problem. In Section V, we describe the experimental setup and discuss the experimental results. We conclude the paper and suggest possible directions for future work in Section VI.

II. Related Work

In recent years, exploiting renewable energy for computing received a lot of attention. Many studies have focused on improving the efficiency of the cooling system in data centers through designing smart facilities, incorporating thermal awareness into server provisioning, and workload management [8], [17], [26]. Some studies have proposed solutions for installing data centers close to the renewable energy sites and alternating cooling supplies [1], [5], [7]. However, it is difficult to only rely on renewable energy due to its unpredictability and variability. Fortunately, short-term carbon intensity can be predicted with high accuracy [9]. Statistics [11], [25] show that there is a wide variation in carbon intensity of energy sources by time and location. This reflects the large potential impact that job scheduling and resource allocation can have on reducing carbon emission.

Shifting workloads towards the time periods that have low carbon intensity has been considered as an effective way of reducing the carbon emissions of data centers [16], [17], [21], [27]. For example, Google's Carbon Intelligent Computing System (CICS) [21] considers the scheduling of batch jobs to jointly minimize energy consumption and carbon footprint. The optimizer has as input the carbon intensity forecast over the next 24 hours and also the prediction of the batch jobs load. It assumes that jobs are not interactive and can be executed throughout the day as long as the computational capacity is preserved. Liu et al., [17] studied shifting flexible workloads to minimize the total operating costs while maximizing the revenue as the objective. They consider several hours to multiple days as the deadline for the jobs. However, these studies in the area of shifting workloads do not consider the schedule of each individual job with its specific requirements when making decision about the execution order.

In contrast to shifting workload, job-based scheduling focuses on the scheduling of each individual job with its requirements. Energy-aware job scheduling and placement has been widely studied in the cloud computing environment [6], [10], [13], [22]. Rocha et al. [22] developed an energy-aware scheduling algorithm in which jobs are assigned to cluster nodes based on their scores for each node. The scores of the jobs for each node is obtained based on the resource requirements of the job and the performance of the node and by using a predication model. Cadorel et al. [10] presented a heuristic algorithm for the workflow scheduling problem with the aim of minimizing the number of active servers. However, these studies did not consider the availability of renewable energy in their decision making. In this paper, we address the Carbon-aware Job Scheduling Problem(CJSP). Each job is specified by its resource requirements, execution time, arrival time, and a deadline. The data center has a limited computational capacity and the carbon intensity of the power supplied to the data center may vary over time. We design an LP-based

algorithm for solving CJPS which is fast and suitable for cloud systems. The algorithm provides approximation bounds on the objective function and the constraints.

III. Carbon-aware Job Scheduling

We address the *Carbon-aware Job Scheduling Problem* (CJSP) for a cloud system. We consider a discrete time slotted system in which a set of jobs $\mathcal{N}$ is to be scheduled on a data center within a time horizon T so that the total carbon footprint generated by executing the jobs is minimized.

Each job $i \in \mathcal{N}$ is characterized by the tuple (a_i, d_i, r_i, l_i), where a_i is the arrival time, d_i is the deadline, r_i is the required amount of the computational resource, and l_i is the time needed for executing job i, all are integers. A unit of time required by job i corresponds to the time for executing unit-time tasks of the job. When we refer to a task, we mean a chunk of computation that can be executed in a unit of time. For the sake of readability, we consider a single type of computational resource, e.g. CPU, but the formulation can be extended to multiple types of resources. We allow interruption of jobs. However, whenever a job is scheduled in a time slot, it must be executed for the entire duration of the slot.

The data center is characterized by the tuple $(B_t, I_t, P^{idle}, P^{max})$, where B_t is the computational capacity and I_t is the carbon intensity in time slot t, and P^{idle} and P^{max} are power consumption parameters of the data center. We assume that the length of each time slot is fixed and can be set to a value that satisfies the QoS for the execution of the job. In the following, we define the constraints for CJSP, the energy consumption model of the data center, and the objective of CJSP. The notation we use in the formulation is provided in Table I.

Scheduling constraints. Each job i requires l_i time slots. These time slots are not necessarily consecutive,

$$\sum_{t \in \{a_i, \ldots, d_i\}} x_{it} = l_i \quad \forall i \in \mathcal{N}, \tag{1}$$

where x_{it} is a binary variable and $x_{it} = 1$ if job i is executed in time slot t and $x_{it} = 0$ otherwise.

Capacity constraints. The total amount of resources requested cannot exceed the available capacity of the data center,

$$\sum_{i \in \mathcal{N}} x_{it} \cdot r_i \leq B_t, \quad \forall t \in \mathcal{T}. \tag{2}$$

Energy consumption model. We consider the linear model presented by Fan et al. [14] for tracking the dynamic energy consumption of the data center. Based on this model, the computational energy consumption in time slot t includes the the idle energy consumption and the dynamic energy consumption. Here, we only consider the dynamic energy consumption of the data center since the idle energy consumption is static and the schedule of jobs does not have any impact on it. The dynamic energy consumption in time slot t is proportional to the CPU utilization and is defined as,

TABLE I: Notation

Notation	Description
T	Time horizon for executing all jobs
$\mathcal{N}$	Set of n jobs
B_t	Computational capacity of the data center in time slot t
P^{idle}	Basic power consumption
P^{max}	Peak power consumption
I_t	Carbon intensity of the electricity of the data center in time slot t
r_i	Computational resource requested by job i
l_i	Duration time of job i
a_i	Arrival time of job i
d_i	Deadline to complete job i
f	Length of each time slot
$\mathcal{T}$	Set of time slots $\{1, \ldots, T\}$
x_{it}	Binary variable, it is 1 if job i is executed in time slot t and 0, otherwise

$$E_t = (P^{max} - P^{idle}) \cdot \frac{\sum_{i \in \mathcal{N}} x_{it} \cdot r_i}{B_t} \cdot f,$$

where P^{idle} is the basic power consumption (Watt), P^{max} is the peak power consumption (Watt), and f is the length of each time slot (second).

Objective. The goal of the CJSP is to minimize the total carbon footprint generated by executing the jobs in the data center. It is given by,

$$\min \quad \sum_{t \in \mathcal{T}} I_t \cdot (P^{max} - P^{idle}) \cdot f \cdot \frac{\sum_{i \in \mathcal{N}} x_{it} \cdot r_i}{B_t}. \tag{3}$$

By defining $c_{it} = \frac{I_t \cdot (P^{max} - P^{idle}) \cdot f \cdot r_i}{B_t}$ and by considering the above constraints, we formulate the CJSP problem as an Integer Linear Program (ILP):

ILP-CJSP

$$\min \quad \sum_{i \in \mathcal{N}} \sum_{t \in \mathcal{T}} c_{it} \cdot x_{it} \tag{4}$$

subject to:

$$\sum_{t \in \{a_i, \ldots, d_i\}} x_{it} = l_i, \quad \forall i \in \mathcal{N}, \tag{5}$$

$$\sum_{i \in \mathcal{N}} x_{it} \cdot r_i \leq B_t, \quad \forall t \in \mathcal{T}, \tag{6}$$

$$x_{it} \in \{0, 1\}, \quad \forall i \in \mathcal{N}, \forall t \in \mathcal{T}. \tag{7}$$

The optimal solution obtained by solving CJSP-MIP will be used in the experimental results section to compare with the solution obtained by the proposed algorithm for solving CJSP.

A. Complexity

We show that CJSP is NP-hard by proving that the special case of the problem is NP-hard. We consider a special case of CJSP (let us call it S-CJSP) in which all jobs arrive at the same time ($a_i = 0$) and they have the same deadline $d_i = T$. We assume that the run time of each job is one time slot ($l_i = 1$) and the capacity of the data center in each time slot is $B_t = B$. Finding a feasible solution for S-CJSP is equivalent

to the decision problem of minimizing the makespan of N jobs on T identical parallel machines in which the duration of each job is r_i. The goal is to find an assignment for the jobs while the maximum completion time on each machine does not exceed B. The decision problem of the makespan minimization problem is known as NP-complete (problem SS8 in [15]). Thus, finding a feasible solution for S-CJSP is NP-complete. This implies that the decision version of S-CJSP is NP-complete. Thus, S-CJSP is NP-hard and CJSP which is the general case of S-CJSP is NP-hard as well.

IV. An approximation algorithm for CJSP

In Section III, we showed that CJSP is an NP-hard problem. Thus, finding an optimal solution for CJSP in reasonable amount of time is infeasible unless $P = NP$. We develop an LP-based approximation algorithm called APX-CJSP to solve CJSP efficiently. We show that if there is a feasible solution for the problem, APX-CJSP can approximate it with strong bounds: for a given set of jobs, APX-CJSP finds a schedule with the total carbon footprint not greater than the optimal value OPT, while the total resources allocated to jobs at each time slot does not exceed $2B_t$ and a maximum of two tasks of a job can be executed in the same time slot.

We observe that when the run time of all jobs is one (i.e., $l_i = 1$), CJSP is an assignment problem in which the goal is to assign each job to a time slot so that the total capacity in each time slot t does not exceed B_t while the total carbon footprint is minimized. When $l_i > 1$, the problem is more complicated. Let us consider each unit of time required to execute a job as one task. Thus, job i has l_i tasks in which no two tasks can be executed at the same time slot. We can consider these tasks as incompatible tasks. Thus, CJSP with $l_i > 1$ is an assignment problem in which conflicting items cannot be assigned to the same time slot. Based on these observations, we develop an LP-Based algorithm which is an extension of an algorithm developed for the Generalized Assignment Problem (GAP) [24].

An algorithmic description of the proposed APX-CJSP algorithm is given in Algorithm 1. The input to APX-CJSP is the information of the set of jobs $\mathcal{N}$ and the data center. The output of the algorithm consists of the scheduling matrix $X = \{x_{it}\}$ and the total carbon footprint CFP. Let us denote by $\bar{x}_{it}$, the solution obtained by solving the LP relaxation of ILP-CJSP in which we replace constraint (7) with $0 \leq x_{it} \leq 1$ (Line 1). Based on this solution, we build a bipartite graph $H = (U, V, E)$, where U and V are two disjoint sets of nodes, and E is the set of edges connecting the nodes in U to nodes in V. Let us call the nodes in U task bins and the nodes in V time slot bins. Along building the graph, we obtain a fractional matching solution for H stored in $\{\bar{z}^{ts}_{ik}\}$ (Line 2). Then, the algorithm finds a complete matching $\{z^{ts}_{ik}\}$ in graph H in which each node in U is exactly assigned to one node in V (Line 3). Then, based on the complete matching solution, the algorithm finds a solution for CJSP (Lines 4-7). The value of x_{it} is defined as the the total number of matching of task bins of job i to the time slot bins of time slot t. In subsection IV-B, we show that the maximum value of x_{it} obtained by the algorithm is two, which means a maximum of two tasks of a job are scheduled at the same time slot. In the following, we explain the details of building bipartite graph H and finding the complete matching solution.

Algorithm 1 APX-CJSP Algorithm

Input: $\mathcal{N} = \{(a_i, d_i, r_i, l_i)\}$: Set of jobs
Input: $\{c_{it}\}$: Carbon footprint generated by jobs
1: $\{\bar{x}_{it}\} \leftarrow$ LP-Solver$(\mathcal{N}, \{c_{it}\})$
2: $(H(U, V, E), \{\bar{z}^{ts}_{ik}\}) \leftarrow$Build-Graph$(\{\bar{x}_{it}\})$
3: $\{z^{ts}_{ik}\} \leftarrow$ Complete-Matching$(H, \{\bar{z}^{ts}_{ik}\})$
4: $q_t \leftarrow \sum_{i \in \mathcal{N}} \bar{x}_{it} \quad \forall t$
5: $x_{it} \leftarrow 0 \quad \forall i, t$
6: **for each** $i \in \mathcal{N}$ **and** $t \in \mathcal{T}$ **do**
7: $\quad x_{it} \leftarrow \sum_{k \in \{1, \ldots l_i\}} \sum_{t \in \mathcal{T}} z^{ts}_{ik}$
8: **end for**
9: $CFP \leftarrow \sum_{t \in \mathcal{T}} \sum_{i \in \mathcal{N}} c_{it} \cdot x_{it}$
Output: X: Scheduling matrix
Output: CFP: Total carbon footprint

For each job i, we consider l_i task bins, and for each time slot t, we consider $q_t = \lceil \sum_{i \in \mathcal{N}} \bar{x}_{it} \rceil$ time slot bins. We construct the bipartite graph H, where one side of the bipartite graph consists of task bins of the jobs $U = \{(1,1), \ldots (1, l_1), \ldots, (N, 1), \ldots (N, l_N)\}$, and the other side consists of the time slot bins $V = \{(1,1), \ldots (1, q_1), \ldots, (T, 1), \ldots (T, q_T)\}$. The capacity of each time slot bin (t, s) is one and the bin is packed by the values of $\bar{x}_{it}, \forall i \in \mathcal{N}$. On the other hand, since each task requires one time slot processing, it can be considered as a bin of capacity one that is packed with the values of $\bar{x}_{it}, \forall t \in \{a_i, \cdots, d_i\}$. We assume that jobs are sorted in descending order of their resource requirement,

$$r_1 \geq r_2 \ldots \geq r_N.$$

Initially, the task bins and the time slot bins are empty. We start with an arbitrary time slot t and find the first job for which $\bar{x}_{it} > 0$. We place $\bar{x}_{it}$ in time slot bin $(t, 1)$. Then, we pick the next job with $\bar{x}_{it} > 0$ and try to put it in $(t, 1)$. If the remaining capacity of bin $(t, 1)$ is less than $\bar{x}_{it}$, we put a fraction of $\bar{x}_{it}$ into time slot bin $(t, 1)$ to make it full, and put the remaining in the next time slot bin $(t, 2)$. We continue this procedure for all time slots until all $\bar{x}_{it}$'s are packed.

We need to also fill the task bins. Let us assume that o^{ts}_i is a fraction of $\bar{x}_{it}$ that is assigned to time slot bin (t, s). To determine how to assign o^{ts}_i to the task bins, we start with the first non-full task bin (i, k) and try to pack o^{ts}_i to the bin. If the remaining capacity of the task bin is less than o^{ts}_i, we put a fraction of o^{ts}_i to the task bin to make it full. Then, we pack the remaining fraction of o^{ts}_i in the next task bin $(i, k+1)$. Let us denote by $\bar{z}^{ts}_{ik}$, the fraction of o^{ts}_i that is assigned to task bin (i, k). If $\bar{z}^{ts}_{ik} > 0$, we add an edge with weight c_{it} from node (i, k) to node (t, s). Later, we will show that $\{\bar{z}^{ts}_{ik}\}$ is a feasible fractional matching for H with

TABLE II: Example: value of parameters

Parameter	Value	Parameter	Value
$\{a_1, a_2\}$	$\{1, 1\}$	$\{I_1, I_2, I_3\}$	$\{1, 4, 2\}$
$\{d_1, d_2\}$	$\{3, 3\}$	f	1
$\{r_1, r_2\}$	$\{4, 2\}$	P^{max}	2
$\{l_1, l_2\}$	$\{2, 1\}$	P^{idle}	1
$\{B_1, B_2, B_3\}$	$\{5, 5, 5\}$		

maximum objective value OPT. According to [18], from the fraction matching solution, we can obtain, in polynomial time, an integer complete matching $\{z_{ik}^{ts}\}$ with objective value not greater than OPT. In the following, first we give an example to illustrate the construction of the bipartite graph. Then, we will discuss the properties of the algorithm.

A. Illustrative Example

Here, we provide a simple example to show how the algorithm works. We consider a problem instance with two jobs that are to be scheduled in the data center over time slots $\mathcal{T} = \{1, 2, 3\}$. The values of parameters are given in Table II.

Based on the values of the parameters, we obtain matrix $C = \{c_{it}\}$ and solve the LP relaxation of ILP-CJSP. Matrix $\bar{X}$ is the solution obtained by the LP-relaxation. Matrices C and $\bar{X}$ are given by,

$$C = \begin{pmatrix} \frac{4}{5} & \frac{16}{5} & \frac{8}{5} \\ \frac{2}{5} & \frac{8}{5} & \frac{4}{5} \end{pmatrix} \text{ and } \bar{X} = \begin{pmatrix} 1 & 0 & 1 \\ \frac{1}{2} & 0 & \frac{1}{2} \end{pmatrix},$$

respectively.

Based on the values of $\{\bar{x}_{it}\}$, time slot $t = 1$ needs two bins because $q_1 = \lceil \bar{x}_{11} + \bar{x}_{21} \rceil = \lceil 1 + \frac{1}{2} \rceil = 2$. Time slot $t = 2$ needs no bin, and time slot $t = 3$ needs two bins. Figure 2a shows the bipartite graph H built for this example. In this figure, the circles show the task bins and the rectangles show the time slot bins. The weight between each task bin (i, k) and time slot bin (t, s) is denoted by w_{ik}^{ts} and is set to $w_{ik}^{ts} = c_{it}$. We start with time slot $t = 1$ and pack the first time slot bin $(1, 1)$ with $\bar{x}_{11} = 1$, at the same time we pack the first task bin of job $i = 1$ with the same value and add an edge with weight $w_{11}^{11} = c_{11} = \frac{4}{5}$ from task bin $(1, 1)$ to time slot bin $(1, 1)$. We set $\bar{z}_{11}^{11} = 1$. Then, we start packing the second time slot bin of $t = 1$ (i.e., $(1, 2)$) with $\bar{x}_{21} = \frac{1}{2}$. Simultaneously, we pack the task bin $(2, 1)$ of job $i = 2$ with the same value. We add an edge with weight $w_{21}^{12} = c_{21} = \frac{2}{5}$ from task bin $(2, 1)$ to time slot bin $(1, 2)$ and set $\bar{z}_{21}^{12} = \frac{1}{2}$. We skip the second time slot since there is no positive value of $\bar{x}_{i2}$. Then, we start packing the bin of time slot $t = 3$. We pack the time slot bin $(3, 1)$ with $\bar{x}_{13} = 1$, at the same time we pack task bin $(1, 2)$ with the same value and add an edge with weight $w_{12}^{31} = c_{13} = \frac{8}{5}$ from task bin $(1, 2)$ to time slot bin $(3, 1)$. We set $\bar{z}_{12}^{31} = 1$. Then, We fill time slot bin $(3, 2)$ with $\bar{x}_{23} = \frac{1}{2}$ and fill the task bin $(2, 1)$ with this value $\bar{z}_{21}^{32} = \frac{1}{2}$. We add an edge with weight $w_{21}^{32} = c_{23} = \frac{4}{5}$ from task bin $(2, 1)$ to time slot bin $(3, 2)$.

We can easily see that $\{\bar{z}_{ik}^{ts}\}$ is a fractional matching for graph H since for each task bin (i, k), the total assignment is exactly one (i.e., $\sum_{t \in \mathcal{T}} \sum_{s \in \{1 \ldots, q_t\}} \bar{z}_{ik}^{ts} = 1$). The total weight of the assignments is,

$$\sum_{i \in \mathcal{N}} \sum_{k \in \{1, \ldots, l_i\}} \sum_{t \in \mathcal{T}} \sum_{s \in \{1 \ldots, q_t\}} \bar{z}_{ik}^{ts} \cdot w_{ik}^{ts} = \frac{15}{5}.$$

As we stated before, according to [18], if a graph has a fractional matching, then, there is a complete matching for the graph with the total weight not greater than the one obtained by the fractional matching. In Figure 2b, we show a complete matching z_{ik}^{ts} for graph H with the total weight $\frac{14}{5}$.

Now, according to Lines (6-7) of Algorithm 1, and based on the values of z_{ik}^{ts}, we obtain the solution for CJSP. The value of the scheduling matrix X is

$$X = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}.$$

However, this solution violates the capacity constraints for time slot $t = 3$ because both jobs $i = 1$ and $i = 2$ are executed in time slot $t = 3$, resulting in a total resource requirement of $r_1 + r_2 = 6$, which is greater than capacity $B = 5$. In the next subsection, we will show that this violation is bounded by a small factor.

B. Properties

Here, we provide approximation bounds for LP-CJSP algorithm. For a given input, we show that if there is a feasible solution, the LP-CJSP can approximate it with strong bounds. LP-CJSP finds a solution of the total carbon footprint not greater than the optimal value OPT, but violates (i) the capacity constraints by allowing each time slot to process jobs for a total capacity of maximum $2B$, and (ii) executing a maximum of two tasks of a job in a single time slot (i.e.,$x_{it} \in \{0, 1, 2\}$).

Lemma 1. *$\{\bar{z}_{ik}^{ts}\}$ obtained by Build-Graph procedure is a fractional complete matching in H with the total weight not greater than OPT.*

Proof. To prove $\{\bar{z}_{ik}^{ts}\}$ is a fractional matching, we need to show that (i) for each time slot bin, the total assignment is at most one, which is obvious since the algorithm considers a capacity of one for each bin and fill it by at most total assignment of one, and (ii) the total assignment on each task bin is exactly one. From the construction of H, it is easy to see that,

$$\sum_{k \in \{1, \ldots, l_i\}} \sum_{t \in \mathcal{T}} \sum_{s \in \{1 \ldots, q_t\}} \bar{z}_{ik}^{ts} = \sum_{t \in \mathcal{T}} \bar{x}_{it} = l_i \quad \forall i \in \mathcal{N}.$$

Since in the algorithm there are l_i task bins and each bin is not packed unless the previous bins are filled, thus the total assignment on each task bin is exactly one.

Now, we show that the total weight of this fractional matching is not greater than OPT. The total weight of the matching is

$$\sum_{i \in \mathcal{N}} \sum_{k \in \{1, \ldots, l_i\}} \sum_{t \in \mathcal{T}} \sum_{s \in \{1 \ldots, q_t\}} \bar{z}_{ik}^{ts} \cdot c_{it}.$$

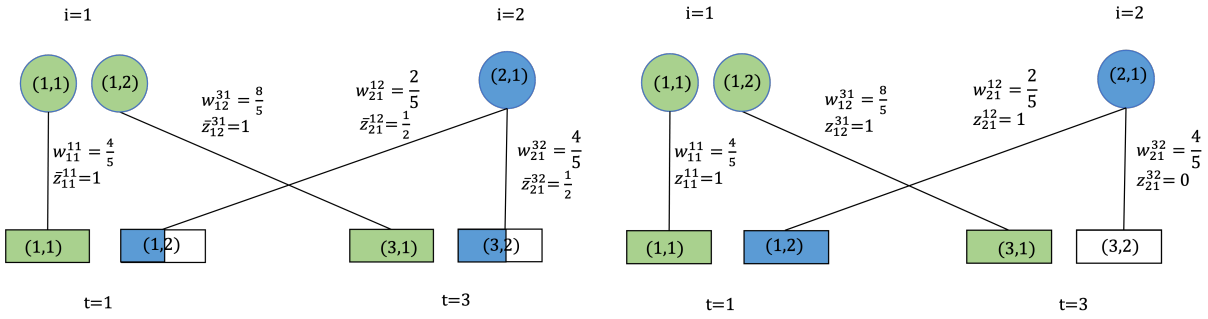


Fig. 2: Example: (a) Bipartite graph H with fractional matching $\bar{z}_{ik}^{ts}$. (b) Bipartite graph H with complete matching z_{ik}^{ts}.

Since $\sum_{k\in\{1,\dots,l_i\}}\sum_{s\in\{1\dots,q_t\}}\bar{z}_{ik}=\bar{x}_{it}$, the total weight of the fractional matching is

$$\sum_{i\in\mathcal{N}}\sum_{k\in\{1,\dots,l_i\}}\sum_{t\in\mathcal{T}}\sum_{s\in\{1\dots,q_t\}}\bar{z}_{ik}^{ts}\cdot c_{it}=\sum_{i}\sum_{t}\bar{x}_{it}\cdot c_{it},$$

which is the objective value obtained by LP relaxation of CJSP. Since the objective value of LP relaxation is a lower bound of the optimal solution, we can conclude that the total weight of the fractional matching is at most OPT. □

Lemma 2. *For a fractional matching $\bar{z}_{ik}^{st}$ in bipartite graph H, there is an integral complete matching z_{ik}^{st} in H with the objective value not greater than that of the fractional matching solution (Theorem 11.1 in [28]).*

Theorem 1. *APX-CJSP algorithm obtains solution $\{x_{it}\}$ with objective value not greater than the optimal value OPT, but in each time slot at most $2B_t$ computational resource is required to complete the jobs and the number of tasks assigned to a single time slot are at most 2 (i.e., $x_{it}\in\{0,1,2\}$).*

Proof. According to Lemma 1 and Lemma 2, z_{ik}^{ts} is a complete matching for graph H with objective value,

$$\sum_{i\in\mathcal{N}}\sum_{k\in\{1,\dots,l_i\}}\sum_{t\in\mathcal{T}}\sum_{s\in\{1\dots,q_t\}}z_{ik}^{ts}\cdot c_{it}\le OPT.$$

On the other hand, according to Line 7 of Algorithm 1, $x_{it}=\sum_{k\in\{1,\dots l_i\}}\sum_{s\in\{1,\dots,q_t\}}z_{ik}^{ts}$. Thus, the total carbon footprint obtained by the algorithm is,

$$\sum_{i\in\mathcal{N}}\sum_{t\in\mathcal{T}}x_{it}\cdot c_{it}=\sum_{i\in\mathcal{N}}\sum_{t\in\mathcal{T}}\sum_{k\in\{1,\dots,l_i\}}\sum_{s\in\{1\dots,q_t\}}z_{ik}^{ts}\le OPT.$$

Now, we discuss the violations of solution $\{x_{it}\}$ on the constraints of ILP-CJS. Consider task bins that can be matched to time slot bin (t,s) (i.e., the task bins with $\bar{z}_{ik}^{ts}>0$). Let $r_{\max(t,s)}$ denote the maximum resource requirement among these tasks. If we consider integral complete matching $\{z_{ik}^{ts}\}$, the total load A_t assigned to time slot t is bounded by,

$$A_t\le\sum_{s=\{1,\dots,q_t\}}r_{\max(t,s)}\quad\forall t\in\mathcal{T}.\tag{8}$$

On the other hand, in the fractional matching $\{\bar{z}_{ik}^{ts}\}$, since we only start the next time slot bin when we have completely fill the previous ones, we will have,

$$\sum_{i\in\mathcal{N}}\sum_{k\in\{1,\dots,l_i\}}\bar{z}_{ik}^{ts}=1\quad\forall t\in\mathcal{T},s\in\{1,\dots,q_t-1\}.\tag{9}$$

Furthermore, since jobs are sorted in decreasing order of their resource requirement,

$$r_{\max(t,s+1)}\le r_i\quad\forall t\in\mathcal{T},s\in\{1,\dots,q_t-1\},\ i\in\mathcal{N},k\in\{1,\dots,l_i\}\ |\bar{z}_{ik}^{ts}>0.\tag{10}$$

Thus, based on Equations (9-10), we will have,

$$r_{\max(t,s+1)}\le\sum_{i\in\mathcal{N}}\sum_{k\in\{1,\dots,l_i\}}\bar{z}_{ik}^{ts}\cdot r_i\quad\forall t\in\mathcal{T},\ s=\{1,\dots,q_t-1\}.\tag{11}$$

Therefore,

$$\begin{aligned}\sum_{s\in\{1,\dots,q_t-1\}}r_{\max(t,s+1)}&\le\sum_{s\in\{1,\dots,q_t-1\}}\sum_{i\in\mathcal{N}}\sum_{k\in\{1,\dots,l_i\}}\bar{z}_{ik}^{ts}\cdot r_i\\&\le\sum_{i\in\mathcal{N}}r_i\cdot\sum_{k\in\{1,\dots,l_i\}}\sum_{s=1\in\{1,\dots,q_t-1\}}\bar{z}_{ik}^{ts}.\end{aligned}$$

Since $\sum_{k\in\{1,\dots,l_i\}}\sum_{s\in\{1,\dots,q_t\}}\bar{z}_{ik}^{ts}=\bar{x}_{it}$, we will have,

$$\sum_{s\in\{1,\dots,q_t-1\}}r_{\max(t,s+1)}\le\sum_{i\in\mathcal{N}}r_i\cdot\bar{x}_{it}.$$

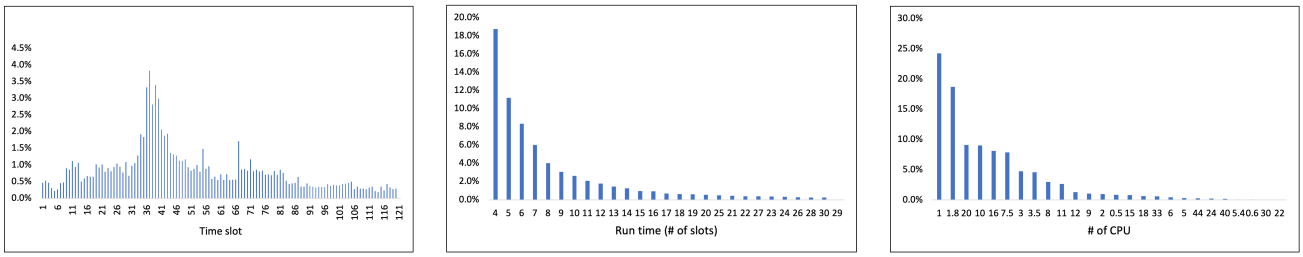
On the other hand, $\bar{x}_{it}$ is a feasible solution for the LP relaxation of CJSP. Thus, we have $\sum_{i\in\mathcal{N}}r_i\cdot\bar{x}_{it}\le B_t$. Consequently,

$$\sum_{s\in\{1,\dots,q_t-1\}}r_{\max(t,s+1)}\le B_t.$$

Furthermore for each job i and each time slot t we assume that $r_i\le B_t$. Thus, $r_{\max(t,s)}\le B_t$ and for the total load on time slot t we will have,

$$A_t\le r_{\max(t,s)}+\sum_{s=1}^{k_t-1}r_{\max(t,s+1)}\le 2B_t.$$

Thus, the maximum resources assigned to a time slot t is bounded by $2B_t$.



(a) (b) (c)

Fig. 3: Workload characteristics distribution: (a) Arrival time. (b) Run time. (c) CPU requirement.

Now, we show that the number of tasks of a job assigned to a time slot is at most 2. For a time slot t and for a job i, the value of $\bar{x}_{it}$ is at most one. Each time slot bin has capacity of 1 and is filled once the previous bins are completely filled. Thus, the value of $\bar{x}_{it}$ is assigned to at most to consequent time slot bins. This means that at most two bins of time slot t may have edge connection to the task bins of job i. Thus, at most two tasks of a job i can be matched to the same time slot.

□

V. Experimental Results

In this section, we investigate the performance of the proposed scheduling algorithm, APX-CJSP, by performing an extensive experimental analysis. We compare the performance of the algorithm against that of the optimal solution obtained by solving ILP-CJSP. We analyze the quality of solutions and the running times of the algorithm for problem instances of different sizes under various settings. In the following, we describe our experimental setup and then analyze the experimental results.

A. Experimental Setup

Computing setup. a batch workload trace recently released by Alibaba [2] to characterize the workload of the data center. The trace is sampled from one of the Alibaba production clusters in a periods of eight days. By analyzing the workload, we observe that more than 91% of the jobs are short, with running time less than 30 minutes. Short-running jobs do not usually tolerate long delay and they are expected to be finished in a timely manner. Thus, their impact on reducing carbon emission may not be high as carbon intensity usually does not change quickly [27]. Thus, in our experiments, we only consider long-running jobs with run time between 30 minutes and 5 hours because they usually have more flexibility on execution time.

We set the length of each time slot to 600 seconds (i.e., $f = 600$). We have investigated various values for the length of time slots; but the behavior of the scheduling algorithm does not change over various values of f. Thus, due to the limited space, we only present the results for $f = 600$. We consider a time horizon of 24 hours (144 time slots) for scheduling jobs that arrived in the first 20 hours (the first 120 time slots). In Figure 3, we plot the characteristics of long-running jobs. Figure 3a shows the histogram of the frequency distribution (%) of arriving jobs in each time slot. Figure 3b shows the the histogram of the frequency distribution (%) of jobs for various run times. Here, we observe that a high percentage of jobs (about 48%) have running time between 30 minutes and 1 hour. Figure 3c shows the histogram of the frequency distribution (%) of jobs for various CPU requirement. Based on these distributions, we generate our workload. In our experimental setup, the number of jobs arriving to the system varies between 100 to 6000. The characteristics of these jobs, such as arrival time, run time, and CPU requirement are determined based on these distributions. The deadline of jobs is calculated based on the slowdown factor α. For example, for job i, the deadline is $d_i = a_i + \alpha * l_i$. In our experimental analysis, we investigate the impact of various values of α on the carbon emission of the data center.

To estimate the power consumption parameters of the data center (i.e., p^{idle}, p^{max}), we consider Gen 2 servers and set $P^{idle} = 480W$, $P^{max} = 1000W$. We get the carbon intensity information of IBM Boulder data center from InfluxDB [3] for a period of 24 hours and use it as the input for the APX-CJSP algorithm.

Performance metrics. The performance of the proposed algorithm is evaluated by considering several performance metrics such as CPU utilization, execution time, CFP ratio, CV ratio, and TAV. In the following, we elaborate on each of these metrics.

CPU utilization in time slot t is denoted by u_t and defined as the ratio between the total resources allocated to the jobs and the capacity of the data center in time slot t,

$$u_t = \frac{\sum_{i \in \mathcal{N}} r_i \cdot x_{it}}{B_t}.$$

What we expect from our scheduling algorithm is to adjust the CPU utilization based on the carbon intensity of the data center.

Capacity violation ratio (CV) is defined as the maximum CPU utilization over all time slots,

$$CV = \max_{t \in \mathcal{T}} \{u_t\}.$$

Our scheduling algorithms guarantees that the value of CV ratio does not exceed two.

CFP ratio is defined as the ratio between the total carbon footprint obtained by AXP-CJSP and the total carbon foot-

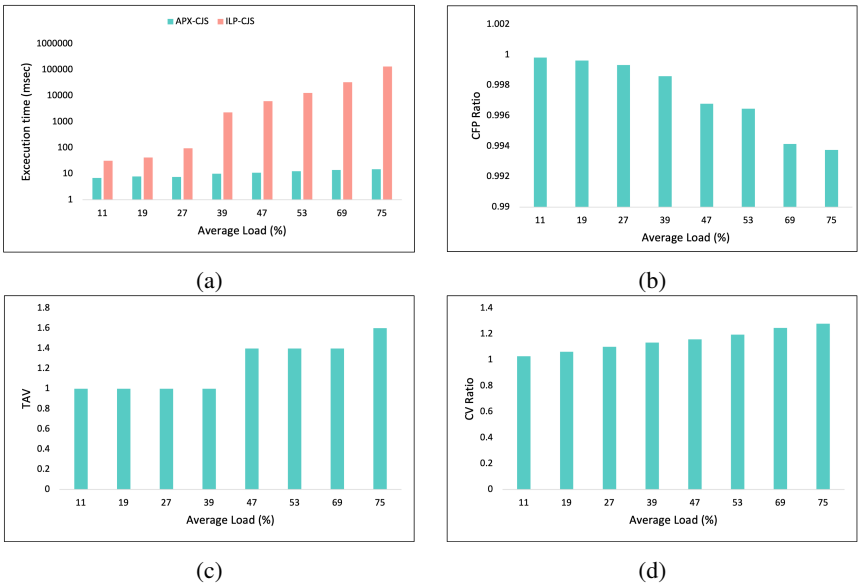


Fig. 4: APX-CJSP vs. ILP-CJSP: (a) Execution time. (b) CFP ratio. (c) TAV. (d) CV ratio.

print obtained by solving ILP-CJSP,

$$CFP = \frac{\sum_{i\in\mathcal{N}}\sum_{t\in\mathcal{T}} c_{it}\cdot x_{it}}{\sum_{i\in\mathcal{N}}\sum_{t\in\mathcal{T}} c_{it}\cdot x^*_{it}},$$

where x^*_{it} is the solution obtained by solving ILP-CJSP and x_{it} is the solution obtained by APX-CJSP.

Task assignment violation or TAV is defined as the maximum number of tasks of a job executed in parallel,

$$\texttt{TAV} = \max_{i\in\mathcal{N},t\in\mathcal{T}}\{x_{it}\}.$$

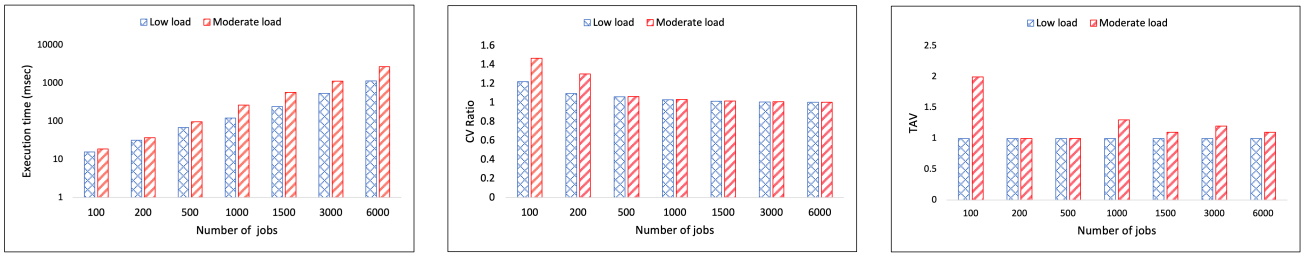
As we stated in problem definition, at most one task of a job can be executed in a time slot. APX-CJSP may violate this constraint; but it guarantees that at most two tasks of the same job are executed in the same time slot.

For each type of problem instance, we execute APX-CJSP and ILP-CJSP algorithms for ten instances. The values of the performance metrics are the average value of the ten executions. APX-CJSP and ILP-CJSP are both implemented in C and the experiments are conducted on an Intel 2.3 GHz Core i9 with 64 GB RAM system. For solving ILP-CJSP and also the LP relaxation of ILP-CJSP, we use the CPLEX 12.8 solver provided by IBM ILOG CPLEX optimization studio [12].

B. Experimental Analysis

We perform an extensive experimental analysis for two classes of problem instances. First, we present the experimental results for the small size instances where each instance consists of 100 jobs. Our goal is to evaluate the performance of APX-CJSP compared to the exact solution obtained by using the CPLEX to solve ILP-CJSP. The second class of instances consists of large size instances in which the number of jobs varies between 100 and 6000. For these instances, CPLEX cannot solve the problem in a reasonable amount of time. Our aim for this class of instances is to analyze the scalability of APX-CJSP.

Small scale instances. Here we analyze the performance of APX-CJSP for a fixed number of jobs. We compare the solutions obtained by APX-CJSP to the exact solution obtained by using the CPLEX to solve ILP-CJSP. Due to the large execution time of CPLEX to obtain the optimal solution for large size instances, here we only consider relatively small size instances. We fix the number of jobs to 100 and consider various capacities for the data center, in such a way that the average load of the data center varies between 11% and 75%. To define the deadline, we consider the same slowdown value for all jobs ($\alpha = 4$). We investigate the impact of the load on the performance of the system. In Figure 4a, we plot the execution times obtained by APX-CJSP and ILP-CJSP using a logarithmic scale. The execution times of ILP-CJSP are several orders of magnitude higher than the execution times of APX-CJSP algorithm for all values of load. The execution times of our proposed algorithm are under 15 millisecond in all cases making it very suitable for deployment in real cloud systems. We observe a slight increase in the execution time of APX-CJSP with the increase of load, but this increase seems reasonable. In fact when the load is higher, the available capacity is more limited and the problem of finding a good solution becomes more complicated. For this reason, both ILP solver and LP solver (which is a part of APX-CJSP algorithm) need more time to obtain a solution. Figure 4b shows the CFP ratio for various values of the load. We observe that in all cases the CFP ratio is less than one (as it is guaranteed by APX-CJSP algorithm). Also in all cases CFP ratio is above 0.99 indicating the total carbon footprint obtained by APX-CJSP is very close to the optimal solution. We also observe that as load increases, CFP ratio decreases. The reason behind this is that, with more restriction on the capacity, obtaining a feasible solution becomes more complicated and APX-CJSP



(a) (b) (c)

Fig. 5: Performance vs. number of jobs: (a) Execution time. (b) CV ratio. (c) TAV.

experiences more violations on task assignment and capacity violation (See Figure 4d, Figure 4c).

Large scale instances. Here we investigate the scalability of APX-CJSP to the number of jobs. We consider two settings for the data center. In the first setting we consider a relatively large capacity (resulting in a low load on the data center), while in the second setting we consider more restricted capacity (resulting in a moderate load on the data center). Under each setting, we investigate the impact of the number of jobs on the performance of APX-CJSP. We vary the number of jobs between 100 to 6000 and adjust the capacity of the data center accordingly to keep the average load at a certain level. For the first setting, we keep the average load around 32%, while in the second setting we keep the average load around 68%. In Figure 5a, we plot the average execution time of APX-CJSP for various number of jobs. We observe that with the increase in the number of jobs, the execution time of the algorithm increases in a polynomial fashion. For example, for the case of $N = 100$, the execution time is about 19 milliseconds while for $n = 6000$, the execution time is about 2,709 milliseconds. Furthermore, we observe that the execution time of APX-CJSP for problem instances with the moderate load is higher than that of the low load. The reason is that with the increase of the load, the problem becomes more complicated and it takes more time for LP-solver to find the optimal fractional solution. Another observation from this figure is that, for all instances, the execution time of the algorithm is less than three seconds, which is negligible compared to the run time requirement of the jobs.

Figure 5b shows the impact of the number of jobs on the CV ratio. We observe that as the number of jobs increases, this ratio decreases. The reason is that packing higher number of jobs in a large capacity is more possible rather than packing smaller number of jobs in a relatively small capacity. Thus, when the capacity is relatively large, a higher percentage of jobs can be packed within the given capacity and a lower capacity violation occurs. Another observation is that in the case of moderate load, the CV violation is higher than that of low load. The reason is that when the load is low, a higher capacity is available for packing the jobs. Thus, a lower violation rate occurs. We observe a similar behaviour for TAV as illustrated in Figure 5c.

Next we analyze the impact of slowdown on the CPU utilization. For this purpose, we consider a large problem instance with 2000 jobs and set the average load to 65%. Figure 6a shows the CPU utilization in each time slot when $\alpha = 4$. We observe that APX-CJS tries to adjust the CPU utilization based on the carbon intensity: the CPU utilization is relatively high when the carbon intensity of the data center is low and the CPU utilization goes down when the carbon intensity is high. However, due to the scheduling and deadline constraints, it cannot always behave in the opposite direction of carbon intensity. Figure 6b shows the CPU utilization for $\alpha = 10$. For this case, we observe that since jobs have more flexibility for execution, the CPU utilization is mostly in the opposite direction of carbon intensity.

VI. Conclusion

In this paper, we proposed an LP-based approximation algorithm for carbon-aware workload scheduling problem in cloud systems. The proposed algorithm guarantees strong approximation bounds on the constraints and the objective function. We performed an extensive experimental analysis to evaluate the performance of the proposed algorithm. The results showed that the solutions obtained by the proposed algorithm are near optimal solutions with small feasibility violations. Furthermore, the small execution time of the algorithm makes it a promising approach for real world cloud systems. In our future research, we plan to develop a framework for addressing both the placement and scheduling of the workloads over multiple geographically distributed data centers. We plan to take into account the uncertainties in the arrival time of the workloads and the carbon intensity of the data centers in the design of the framework.

References

[1] https://www.cultofmac.com/191838/apple-expands-n-c-solar-farm-to-make-data-center-use-100-renewable-energy . Accessed: 2022-02-21.

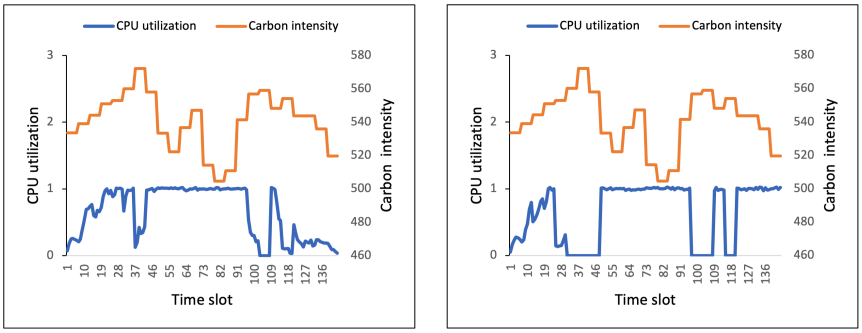
[2] Alibaba inc. 2018. alibaba production cluster data v2018. https://github.com/alibaba/clusterdata/tree/v2018 .

[3] Influxdb. https://www.influxdata.com .

[4] U.s. energy information administration. https://www.eia.gov/. Accessed: 2022-01-30.

[5] Yahoo! compute coop: Next generation passive cooling design for data centers. https://www.energy.gov/eere/amo/yahoo-compute-coop-next-generation-passive-cooling-design-data-centers . Accessed: 2022-02-21.

[6] Z. Abbasi, G. Varsamopoulos, and S. K. S. Gupta. Tacoma: Server and workload management in internet data centers considering cooling-computing power trade-off and energy proportionality. *ACM Trans. Archit. Code Optim.*, 9(2), June 2012.



(a) (b)

Fig. 6: CPU utilization vs. slowdown: (a) $\alpha = 4$. (b) $\alpha = 10$.

[7] S. Akoush, R. Sohan, A. Rice, A. W. Moore, and A. Hopper. Free lunch: Exploiting renewable energy for computing. In *13th Workshop on Hot Topics in Operating Systems (HotOS XIII)*, 2011.

[8] C. Bash, C. D. Patel, and R. K. Sharma. Dynamic thermal management of air cooled data centers. In *Thermal and Thermomechanical Proceedings 10th Intersociety Conference on Phenomena in Electronics Systems, 2006. ITHERM 2006.*, pages 8–pp. IEEE, 2006.

[9] N. D. Bokde, B. Tranberg, and G. B. Andresen. Short-term co2 emissions forecasting based on decomposition approaches and its impact on electricity market scheduling. *Applied Energy*, 281:116061, 2021.

[10] E. Cadorel, H. Coullon, and J. Menaud. A workflow scheduling deadline-based heuristic for energy optimization in cloud. In *2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 719–728, 2019.

[11] D. S. Callaway, M. Fowlie, and G. McCormick. Location, location, location: The variable value of renewable energy and demand-side efficiency resources. *Journal of the Association of Environmental and Resource Economists*, 5(1):39–75, 2018.

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

[13] Z. Dong, W. Zhuang, and R. Rojas-Cessa. Delayed best-fit task scheduling to reduce energy consumption in cloud data centers. In *2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 729–736, 2019.

[14] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. *ACM SIGARCH computer architecture news*, 35(2):13–23, 2007.

[15] M. R. Garey and D. S. Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

[16] L. Lin, V. M. Zavala, and A. A. Chien. Evaluating coupling models for cloud datacenters and power grids. In *Proceedings of the Twelfth ACM International Conference on Future Energy Systems*, e-Energy '21, pages 171–184, New York, NY, USA, 2021. Association for Computing Machinery.

[17] 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.

[18] L. Lovász and M. D. Plummer. *Matching theory*, volume 367. American Mathematical Soc., 2009.

[19] E. Masanet, A. Shehabi, N. Lei, S. Smith, and J. Koomey. Recalibrating global data center energy-use estimates. *Science*, 367(6481):984–986, 2020.

[20] X. Qing and Y. Niu. Hourly day-ahead solar irradiance prediction using weather forecasts by lstm. *Energy*, 148:461–468, 2018.

[21] 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.

[22] 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, 2019.

[23] N. Sharma, P. Sharma, D. Irwin, and P. Shenoy. Predicting solar generation from weather forecasts using machine learning. In *2011 IEEE international conference on smart grid communications (SmartGridComm)*, pages 528–533. IEEE, 2011.

[24] D. B. Shmoys and É. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical programming*, 62(1):461–474, 1993.

[25] M. P. Thind, E. J. Wilson, I. L. Azevedo, and J. D. Marshall. Marginal emissions factors for electricity generation in the midcontinent iso. *Environmental science & technology*, 51(24):14445–14452, 2017.

[26] Z. Wang, A. McReynolds, C. Felix, C. Bash, C. Hoover, M. Beitelmal, and R. Shih. Kratos: Automated management of cooling capacity in data centers with adaptive vent tiles. In *ASME International Mechanical Engineering Congress and Exposition*, volume 43833, pages 269–278, 2009.

[27] P. Wiesner, I. Behnke, D. Scheinert, K. Gontarska, and L. Thamsen. *Let's Wait Awhile: How Temporal Workload Shifting Can Reduce Carbon Emissions in the Cloud*, pages 260–272. Association for Computing Machinery, New York, NY, USA, 2021.

[28] D. P. Williamson and D. B. Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.