Workload Placement with Bounded Slowdown in Disaggregated Datacenters

Amirhossein Sefati*, Mahdi Dolati†, and Majid Ghaderi*
*Department of Computer Science, University of Calgary, Calgary, Canada.
†School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran.
Emails: {amirhossein.sefati, mghaderi}@ucalgary.ca, m.dolati@ipm.ir

***Abstract*—Disaggregated Data Center (DDC) is a modern datacenter architecture that decouples hardware resources from monolithic servers into pools of resources that can be dynamically composed to match diverse workload requirements. While disaggregation improves resource utilization, it could negatively impact workload slowdown due to the latency of accessing disaggregated resources over the datacenter network. To this end, we consider CPU and memory disaggregation and conduct measurements to experimentally profile several popular datacenter workloads in order to characterize the impact of disaggregation on workload execution slowdown. We then develop a workload placement algorithm, called Iterative Rounding-based Placement (IRoP), that given a set of workloads, determines where to place each workload (*i.e.*, on which CPU) and how much local and remote memory allocate to it. The key insight in designing IRoP is that the impact of remote memory latency on slowdown can be substantially masked by assigning workloads to higher-performing CPUs, albeit at the cost of higher energy consumption. As such, IRoP aims to find a workload placement that minimizes the DDC energy consumption while respecting a bounded slowdown for each workload. We provide extensive simulation results to demonstrate the flexibility of IRoP in providing a wide range of trade-offs between energy consumption and workload slowdown. We also compare IRoP with several existing baselines. Our results indicate that IRoP can reduce energy consumption and slowdown in the considered scenarios by up to 8% and 12%, respectively.**

I. Introduction

Motivation. Today's data centers (DCs) are designed based on a server-centric model. The building block of this model is a monolithic server that includes all necessary hardware resources such as CPU, RAM, and NIC to run typical datacenter workloads. One of the main limitations of the server-centric model is the difficulty to achieve full resource utilization due to *resource stranding*, where a server that has used up one type of resource cannot run more workloads even though it may still have large amounts of other resources available. Indeed, measurements in production DCs show that the average utilization of hardware resources is relatively low. For example, a recent report [1] reveals that 80 percent of the time datacenter clusters utilize 10–30% of their CPU capacities and more than half of the time the average memory utilization is around 50%. Beyond the cost of hardware, under-utilization of resources also results in elevated energy consumption, as static energy consumption of fixed-ratio servers in a datacenter is significant [2], which has financial and environmental consequences. Addressing the stranded resource problem has become even more critical as the model of computing is evolving in response to emerging data-centric workloads such as those in ML/AI. These workloads require large amounts of processing capacity as well as memory to work efficiently. Simply over-provisioning servers with more hardware resources not only exacerbates the stranded resource problem but faces practical limitations (*i.e.*, limited DIMM slots for memory on the server board).

To overcome the limitations of the server-centric model, a *resource-centric* model is proposed for datacenters based on resource disaggregation. In a disaggregated datacenter (DDC), server hardware resources are physically or logically disaggregated into homogeneous resource pools from which resources can be allocated to workloads on-demand. The scale of disaggregation can be within a single rack, a group of racks (*i.e.*, a cluster), or the entire datacenter. One of the most crucial components of a disaggregated architecture at any scale is the network that interconnects resource pools. Recent advances in low-latency networking demonstrate that microsecond-scale host-to-host latency is achievable in datacenters [3], with sub-microsecond latency within the host networking stack [4]. Nevertheless, even such network latencies are still high when accessing remote disaggregated memory, noting that accessing local memory on the same server box takes in the order of tens of nanoseconds [5]. As memory access latency increases, execution slows down for those workloads that are memory intensive. To compensate for the slower memory access, workloads can be assigned to higher performing CPUs, *e.g.*, CPUs with higher compute capacity or higher clock frequency. However, this strategy increases datacenter energy consumption (the higher the frequency, the higher the energy consumption) and infrastructure costs.

Different workloads have different levels of sensitivity to memory access latency. In Fig. 1, we have plotted measured *slowdown* for several popular datacenter workloads (see Section II-B for details). The slowdown of a workload is defined as the ratio between its completion times when executed in a disaggregated DC and when executed in a traditional DC. The figure clearly shows that some workloads such as Kmeans [6] are highly sensitive to memory access latency, while others such as WordCount [7] show negligible sensitivity. This behavior can be exploited to minimize the impact of memory disaggregation on workload slowdown. Recently, a few works have considered optimizing workload slowdown through runtime management [8], [9]. But runtime

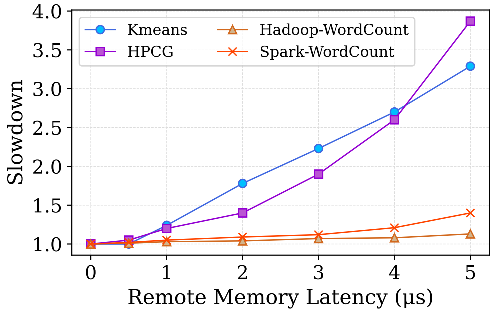


Fig. 1: Workload slowdown as remote memory access latency increases. The local memory ration is fixed at 25%. The x-axis shows the amount of additional latency when accessing remote memory compared to local memory.

management alone is not sufficient as it cannot help during the startup time. For example, workloads with a low ratio of hot to cold pages severely suffer from insufficient low-latency memory at startup [10]. Moreover, when the starting resource configuration of a workload is far from optimal, it would suffer from slowdown while the runtime management tries to transition the workload to an optimal resource configuration. As such, in addition to runtime management, careful workload placement at the time of deployment is needed to ensure workload slowdown is not unacceptably degraded due to remote memory access, while minimizing datacenter energy consumption. The workload placement determines: i) on which CPU to run the workload, and ii) how much local and remote memory allocate to it. Several works have considered workload placement in DDC. However, these works focus on either optimizing workload slowdown [3], [11], [12] or minimizing datacenter energy consumption [2], [13]–[17], without considering the trade-off between the two. In this paper, we aim to study workload placement with bounded slowdown, *i.e.*, guaranteeing slowdown does not exceed a pre-specified target level based on SLAs, while minimizing DDC energy consumption.

Our Work. We consider CPU and memory disaggregation in DC, where each CPU is provisioned with a fixed (small) amount of local memory, but can access remote memory modules, which are considerably larger, over the datacenter network. We assume resource disaggregation is at cluster scale, although our placement algorithm is independent of the aggregation scale. Our first objective is to devise a technique to accurately profile workloads in order to model the impact of remote memory latency and CPU processing capacity on slowdown. To this end, we consider several popular workloads and run each of them in isolation on a modified Linux system and measure its completion time. The modified Linux system allows us to inject arbitrary amount of delay when accessing remote memory (emulated using the swap space), change the ratio of remote and local memory, and scale the CPU frequency dynamically. We then approximate the impact of remote memory latency and CPU frequency on slowdown using piece-wise linear functions. Our second objective is to design a fast algorithm for energy-efficient workload placement with bounded slowdown. To this end, we use the constructed workload profiles and formulate the workload placement problem as a mixed-integer linear problem (MILP). The MILP formulation is NP-hard, and thus we focus on designing a fast polynomial time algorithm to solve it called IRoP. The design of IRoP is based on transforming the MILP problem into a modified variant of the multi-dimensional bin packing problem [18] and applying the framework of deterministic rounding of relaxed integer programs. We mathematically analyze IRoP and derive its theoretical approximation ratio and runtime complexity.

The main contributions of this paper are:

- We present a framework for profiling workload slowdown with respect to remote memory and CPU processing capacity using simple piece-wise linear functions. We empirically show that the linear functions provide highly accurate estimation of real-world workload slowdown.
- We formulate workload placement in DDC as an MILP. We then design an efficient approximation algorithm, called IRoP, to solve the MILP based on the deterministic rounding framework and analyze its theoretical performance.
- We conduct extensive experiments to evaluate the performance and utility of IRoP in terms of energy consumption and workload slowdown in a variety of realistic scenarios. Our results indicate that IRoP reduces both energy consumption and slowdown in the considered scenarios by up to 8% and 12%, respectively.

Organization. Section II describes the proposed disaggregation and workload models. Sections III and IV present the workload placement problem formulation and proposed algorithm, respectively. Evaluation results are presented in Section V. We review related works in Section VI while Section VII concludes the paper.

II. Profiling Datacenter Workloads

In this section, first we present the datacenter model considered in our work and then focus on developing workload profiles that succinctly capture the impact of resource disaggregation on slowdown. Table II summarizes the main notations used throughout the paper.

A. Datacenter Model

Disaggregation Model. We consider a physically disaggregated datacenter consisting of two types of modules. The first type, denoted by $\mathcal{C}$, encompasses modules that provide computing capacity. Each module $c \in \mathcal{C}$ is equipped with N_c CPU cores operating at a maximum frequency of F_c, which determines their processing capacity. Additionally, each module $c \in \mathcal{C}$ is equipped with L_c amount of local memory (*i.e.*, DRAM). This local memory is accessible only by the CPU cores within the same module. The second type of modules exclusively offer memory capacity and can be accessed by any CPU core in the datacenter via the datacenter network. The

TABLE I: Profiled Workloads.

Workload	Implementation	Dataset Size	Memory Usage
Kmeans	Python-TensorFlow [6]	200MB	2GB
WordCount	Java SE [7]	10GB	8GB
HPCG	C++ [19]	1GB	12GB

total memory capacity provided by remote memory modules is denoted by M.

Energy Consumption Model. Previous studies have indicated that servers in a datacenter account for 85% of the overall energy consumption [17]. Among the server components, CPU modules contribute to 85%-88% of the total energy consumption [17]. Therefore, we focus on CPU modules as the primary energy consumers. The energy consumption of CPU module c is modeled as:

$$E_c(x, y) = I_c^s \cdot x + I_c^d \cdot y, \tag{1}$$

where x is an indicator variable that equals zero if module c is powered off and one otherwise. The variable y represents the utilization (*i.e.,* load) of the module. Note that when $x = 0$, y must also equal zero. Coefficient I_c^s represents the static energy consumption when the module is powered on, regardless of the load. Coefficient I_c^d denotes the maximum dynamic energy consumption of CPU module c at full load. The maximum energy consumption of the module, when operating at full load, is given by $E_c(1, 1) = I_c^s + I_c^d$. The static energy consumption is typically around 75% of the maximum [17].

Workload Model. We consider a batch of workloads denoted by $\mathcal{W}$. Each workload $w \in \mathcal{W}$ is characterized by a tuple $\langle \nu_w, \phi_w, \mu_w, \Delta_w \rangle$, where ν_w denotes the requested number of CPU cores, ϕ_w is the requested frequency of CPU cores, and μ_w represents the total amount of requested memory. We assume that the DDC provides incentives for workloads that are willing to tolerate some level of slowdown, *e.g.,* lower pricing. Using the parameter Δ_w, each workload can specify how much slowdown it is willing to tolerate as part of its SLA.

B. Slowdown Characterization

We assume that workload w can function with CPU cores whose frequencies are different from ϕ_w and varying amounts of local and remote memory as long as the total memory received is μ_w. Modern operating systems have mechanisms to transparently deal with heterogeneous memory latency through the support for Non-Uniform Memory Access (NUMA).

We demonstrate that the DDC operator can proactively generate profiles for popular workloads within the DDC in an offline manner, as we have done the same in this paper. This proactive profile-building approach is specifically relevant for large-scale workloads that benefit from disaggregated memory. Alternatively, a passive profile construction strategy can be employed, leveraging runtime management. Under this approach, when a workload arrives at the DDC without an existing profile, it is treated with zero tolerance during the workload placement phase. Subsequently, the runtime management system attempts to identify an appropriate resource configuration for the workload while concurrently constructing a profile during the process.

TABLE II: Important Notations.

Datacenter Notations	
Symbol	Definition
$\mathcal{C}$	Set of all computation-cable modules
F_c	Frequency of CPU cores in module c
N_c	Number of CPU cores in module c
L_c	Amount of local memory in module c
I_c^s	Static energy consumption of module c
I_c^d	Dynamic energy consumption of module c
M	Total amount of remote memory in DDC
Workload Notations	
Symbol	Definition
$\mathcal{W}$	Set of all workloads
ν_w	Number of cores requested by workload w
ϕ_w	Processor frequency requested by workload w
μ_w	Total amount of memory requested by workload w
$P_w^f(.)$	Model to show the effect of frequency on slowdown of workload w
$P_w^m(.)$	Model to show the effect of memory on slowdown of workload w
Δ_w	Maximum acceptable slowdown of workload w

To characterize the effect of remote memory and CPU frequency on the slowdown of workloads, we profile the behavior of three datacenter workloads that have an iterative approach which requires multiple accesses to the memory: (1) K-means clustering [6], (2) WordCount on a sizable 10 GB dataset obtained from web crawling of Project Gutenberg [7], and (3) High-Performance Conjugate Gradient (HPCG) benchmark [19]. Table I provides further information about the mentioned workloads. The profiling was conducted on a computer equipped with 16 GB of RAM, an Intel(R) Core(TM) i9-12700 processor running at 2.4 GHz, and operating Linux Ubuntu 22.04 LTS.

Effect of Remote Memory. To characterize the impact of memory, we perform a set of experiments where we change the ratio of remote to local memory for each workload. To simulate local and remote memories, we utilize *Ramdisk* in Linux, which allows designating a portion of local memory as a disk representing remote memory. Additionally, we modified the page swap procedure in the Linux Kernel version 6.1 to inject an artificial delay to major page swaps to emulate the network latency experienced when accessing remote memory in DDC. Finally, while a 5μs delay is artificially injected as the remote memory latency, we measure the completion time of each workload using *time* library in the Linux Kernel. For each workload, the completion time has been measured 10 times and the averaged results are presented in Fig. 2. As expected, by increasing the ratio of remote memory from zero to 75%, the slowdown also increases. However, workloads exhibit different levels of sensitivity to remote memory. Specifically, K-Means and HPCG experience slowdowns of approximately 4x and 3.5x, respectively, while variations of WordCount encounter less than a 1.5x slowdown. The plot indicates that a piece-wise linear function is a suitable choice for modeling the observed slowdowns. To validate this hypothesis, we fitted piece-wise functions with one, two, and three segments to the

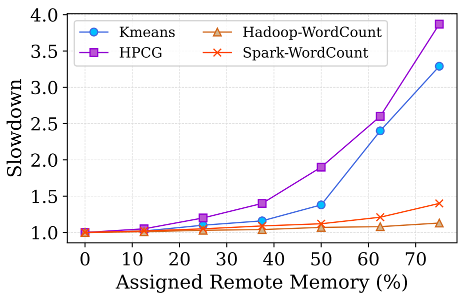


Fig. 2: Workload slowdown as the assigned remote memory ratio increases. The injected latency for remote memory is fixed at 5μs.

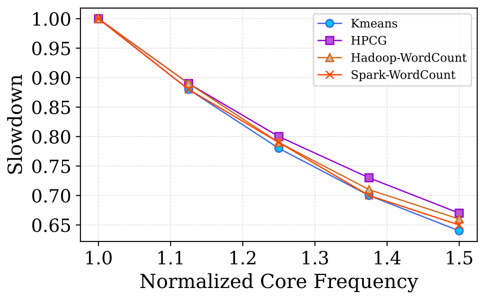


Fig. 3: Workload slowdown as the assigned CPU frequency increases. Lower slowdown means faster execution.

data points of each workload, which indicated that a three-piece function yields accurate estimation. Table III showcases the Mean Squared Errors (MSE) for the three-piece function, including the average, minimum, and maximum values.

TABLE III: Mean Squared Error (MSE) of Linear Models.

Workload	Remote Memory			CPU Frequency		
	Min	Avg	Max	Min	Avg	Max
Kmeans	10^{-7}	1.7	4.3	0.8	1.7	2.4
HPCG	10^{-8}	0.9	3.3	1.0	1.6	2.0
Spark-WordCount	10^{-9}	0.09	0.3	0.9	1.5	2.1
Hadoop-WordCount	10^{-8}	0.2	0.6	0.8	1.6	2.0

Let $P_w^m(r)$ denote the piece-wise linear approximation of the impact of remote memory on workload slowdown. We enforce a minimum of 25% local memory allocation to mitigate significant workload slowdown similar to [5], [20]. Utilizing three segments with breakpoints at 25%, 50%, and 75% for the ratio of remote memory, $P_w^m(r)$ is expressed as:

$$P_w^m(r) = \begin{cases} a_1 \cdot r + c_1 & 0 \leqslant r \leqslant 0.25 \\ a_2 \cdot r + c_2 & 0.25 \leqslant r \leqslant 0.5 \\ a_3 \cdot r + c_3 & 0.5 \leqslant r \leqslant 0.75, \end{cases} \tag{2}$$

where, r represents the percentage of total requested memory allocated remotely for the workload, while a_i and c_i are workload-specific coefficients obtained from profiling. When no remote memory is assigned to a workload ($r = 0$), we anticipate zero slowdown, allowing us to to set c_1 to zero.

Effect of CPU Frequency. To improve the slowdown of workloads, it is possible to assign a CPU module with a higher frequency rate than the workload originally requested. To characterize the impact of CPU frequency on the slowdown of workloads, we conducted experiments by measuring the completion time of all workloads at different CPU frequencies. We used the Linux Kernel module *cpupower* to adjust the core frequencies, ranging from the base frequency of 1.6 GHz to the maximum of 2.4 GHz. Fig. 3 illustrates that the normalized slowdown of workloads decreases approximately linearly as the normalized frequency increases from 1.0 to 1.5. For each workload, we fitted a linear function to the data points and calculated the MSE of the linear approximation, which is presented in Table III. Based on the results, using a linear function to model the effect of CPU capacity on workload slowdown provides a reasonable level of accuracy. Therefore, we consider the function $P_w^f(x)$ defined as following to measure the effect of frequency on workload w:

$$P_w^f(x) = \frac{\phi_w}{x}, \tag{3}$$

where, ϕ_w represents the requested CPU frequency by workload w, while x denotes the received frequency.

III. Workload Placement Problem

In this section, we formally define the problem of Disaggregated Workload Placement (DWLP) and formulate it as an optimization problem, presented in Problem 1. Table IV provides the decision variables used in the formulation.

A. Workload Placement

Let $\mathcal{W}$ denote the set of workloads that are planned for deployment. We introduce the binary decision variable $y_{w,c}$ to indicate whether workload w is assigned to CPU module c. Previous studies such as [12], [17], [21], [22] have demonstrated that current networking technologies cannot fulfill the network requirements for CPU-to-CPU communications. Therefore, in DWLP, workloads are assigned to a single CPU module with a sufficient number of cores. Constraint (4b) ensures that each workload is assigned to exactly one CPU module.

TABLE IV: Decision Variables.

Symbol	Definition
z_c	Activation of module c
$y_{w,c}$	Assignment of workload w to module c
x_w	Fraction of local to total requested memory of workload w
f_w	Frequency of allocated CPU to workload w
u_c	CPU utilization of module c
$\widetilde{N}_c$	Number of allocated cores of module c
$\widetilde{L}_c$	Amount of allocated local memory of module c

Let the decision variable $x_w \in [0, 1]$ represent the proportion of the memory requirement of workload w that is allocated locally. Constraints (4c)-(4f) ensure that the processing and (local) memory loads of all assigned workloads to a CPU module comply with its capacity. Here, z_c is an indicator variable that shows whether processing module $c \in \mathcal{C}$ is on or off. A workload can only be assigned to powered on modules. Recall that ν_w and μ_w denote the number of CPU

Problem 1: DWLP: Disaggregated Workload Placement.

$$\text{Min. } (1-\eta)\cdot\sum_{w\in\mathcal{W}}\theta_w+\eta\cdot\sum_{c\in\mathcal{C}}E_c(z_c,u_c) \tag{4a}$$

$$\text{s.t. } \sum_{c\in\mathcal{C}}y_{w,c}=1, \qquad \forall w\in\mathcal{W} \tag{4b}$$

$$\sum_{w\in\mathcal{W}}y_{w,c}\cdot\nu_w\leqslant\widetilde{N}_c, \qquad \forall c\in\mathcal{C} \tag{4c}$$

$$\sum_{w\in\mathcal{W}}y_{w,c}\cdot x_w\cdot\mu_w\leqslant\widetilde{L}_c, \qquad \forall c\in\mathcal{C} \tag{4d}$$

$$\widetilde{N}_c\leqslant N_c\cdot z_c, \qquad \forall c\in\mathcal{C} \tag{4e}$$

$$\widetilde{L}_c\leqslant L_c\cdot z_c, \qquad \forall c\in\mathcal{C} \tag{4f}$$

$$\sum_{w\in\mathcal{W}}(1-x_w)\cdot\mu_w\leqslant M, \tag{4g}$$

$$0.25\leqslant x_w, \qquad \forall w\in\mathcal{W} \tag{4h}$$

$$\theta_w=\sum_{c\in\mathcal{C}}y_{w,c}\big(P^m_w(1-x_w)\cdot P^f_w(F_c)\big), \forall w\in\mathcal{W} \tag{4i}$$

$$\theta_w\leqslant\Delta_w, \qquad \forall w\in\mathcal{W} \tag{4j}$$

$$u_c=\frac{\widetilde{N}_c}{N_c}, \qquad \forall c\in\mathcal{C} \tag{4k}$$

$$z_c\in\{0,1\}, y_{w,c}\in\{0,1\}, x_w\leqslant 1\,. \tag{4l}$$

cores and the total amount of memory requested by workload w, respectively. The constraint in (4e) ensures that the total number of cores assigned to the workloads in processing module $c\in\mathcal{C}$ does not exceed its total number of cores. Equation (4f) serves a similar purpose, but for the local memory capacity of processing module $c\in\mathcal{C}$. Constraint (4g) ensures that the total remote memory assigned to workloads does not exceed the capacity of remote memory modules.

B. Workload Slowdown

Workload slowdown is impacted by remote memory access latency and allocated CPU frequency. First, as discussed in Subsection II-B, we ensure that each workload receives at least 25% of the requested memory locally with Constraint (4h). We denote the total change in slowdown of workload w as θ_w, which is computed in Constraint (4i). Functions $P^m_w(\cdot)$ and $P^f_w(\cdot)$ in Constraint (4i) are defined in (2) and (3), respectively. Specifically, $P^m_w(1-x_w)$ represents the slowdown of workload $w\in\mathcal{W}$ if $(1-x_w)$ fraction of its total requested memory is allocated from the remote memory modules. Similarly, $P^f_w(F_c)$ is the slowdown (or speedup) caused by assigning a CPU module with a processing capacity of F_c to a workload that originally requested a processing capacity of ϕ_w. We assume that each workload has a maximum acceptable slowdown, denoted by Δ_w. Constraint (4j) ensures that the final slowdown of each workload is not worse than Δ_w.

C. Energy Consumption

To model the dynamic energy consumption of CPU modules, we use the utilization of CPUs, which is defined as the ratio of active cores in a module to the total number of available cores in that module. The utilization of processing module $c\in\mathcal{C}$ is given by Constraint (4k). Following that, the energy consumption of module c is computed by applying (1).

Problem 2: F-DWLP: DWLP with Fixed Memory Ratios.

$$\text{Min. } (1-\eta)\cdot\sum_{w\in\mathcal{W}}\theta_w+\eta\cdot\sum_{c\in\mathcal{C}}E_c(z_c,u_c) \tag{5a}$$

s.t. (4b), (4c), (4e), (4f), (4k)

$$\sum_{w\in\mathcal{W}}y_{w,c}\cdot x_{w,m}\cdot\mu_w\leqslant\widetilde{L}_c, \qquad \forall c\in\mathcal{C} \tag{5b}$$

$$\sum_{w\in\mathcal{W}}\sum_{c\in\mathcal{C}}y_{w,c}\cdot(1-x_{w,c})\cdot\mu_w\leqslant M, \tag{5c}$$

$$\theta_w=\sum_{c\in\mathcal{C}}y_{w,c}(P^m_w(1-x_{w,c})\cdot P^f_w(F_c)), \forall w\in\mathcal{W} \tag{5d}$$

D. Optimization Objective

The objective is defined in (4a). The objective has two components. The first component is the total slowdown of workloads, and the second component defines the total energy consumed by the active CPU modules. By increasing the total energy consumption (*i.e.,* due to allocating higher capacity CPUs), it is possible to decrease the total workload slowdown, and vice versa. Therefore, there is a trade-off between these two components. We use coefficient $0\leqslant\eta\leqslant 1$ to adjust the relative importance of these components. Notice that, as per constraint (4j), the maximum slowdown of each workload is guaranteed per SLA requirements. The reason we have included total slowdown in the objective is to allow the workload placement algorithm to provide even lower slowdown to the workloads if doing so does not disproportionately affects the energy costs of the DDC. This is feasible, for example, when energy consumption is less important than slowdown.

IV. Proposed Algorithm

The problem DWLP involves a joint allocation of memory and CPU frequency, which makes it hard to solve. Our key idea to simplify the problem is based on the fact that every workload's slowdown requirement is satisfied when its achieved slowdown is exactly Δ_j. Using this idea, we expect near-optimum results while we transform DWLP into a simpler problem, called *DWLP with Fixed Memory Ratios* (F-DWLP) and focus on solving this problem as a proxy for the original DWLP problem. However, while F-DWLP is simpler than DWLP, it is still an extension of the multi-dimensional bin-packing problem [18], which is a well-known NP-complete problem. Therefore, we design an approximation algorithm, called *Iterative Rounding-based Placement* (IRoP), using the deterministic rounding framework [23], to obtain a solution for the placement problem in polynomial time.

A. Fixing Memory Ratios

To transform DWLP into F-DWLP, we compute the minimum local memory ratio for each workload w and for each computing module c that is sufficient to satisfy the slowdown constraint of the workload under any CPU allocation strategy determined by $y_{w,c}$. This ratio is computed as follows:

$$x_{w,c}\leftarrow\max\left\{1-(P^m_w)^{-1}\Big(\frac{\Delta_w}{P^f_w(F_c)}\Big),0.25\right\}, \tag{6}$$

Algorithm 1: IRoP: Iterative Rounding-based Placement.

```
   Input  : $\mathcal{M}$: Instance of F-DWLP
   Output: $\{z_c, y_{w,c}\}$: Activation and Assignment Decisions
1  $\widetilde{\mathcal{M}} \leftarrow$ relax($\mathcal{M}$)
2  $\{\widetilde{z}_c, \widetilde{y}_{w,c}\} \leftarrow$ solve($\widetilde{\mathcal{M}}$)
3  for $w \in \mathcal{W}$ do
4      $\overline{\mathcal{C}}_w \leftarrow \{\}$
5      while $\overline{\mathcal{C}}_w \neq \mathcal{C}$ do
6          $c' \leftarrow \arg\max_{c \in \mathcal{C} - \overline{\mathcal{C}}_w} \widetilde{y}_{w,c}$
7          $y_{w,c'} \leftarrow 1$
8          $\{\widetilde{z}_c, \widetilde{y}_{w,c}\} \leftarrow$ solve($\widetilde{\mathcal{M}}$)
9          if $\widetilde{\mathcal{M}}$ is feasible then
10             break
11         else
12             $y_{w,c'} \leftarrow 0$
13             $\overline{\mathcal{C}}_w$.add($c'$)
14     if $|\overline{\mathcal{C}}_w| = |\mathcal{C}|$ then
15         return FAIL
16 return $\{z_c, y_{w,c}\}$
```

where, the computation of the inverse of the memory slowdown function, *i.e.*, $(P^m_w)^{-1}(\cdot)$, is straightforward as the function is linear. Notice that we ensure the allocated local memory ratio is not less than 25%, which was mandated in Constraint (4h) in DWLP. Considering minimum local memory assignment retains a subset of placement options where the slowdown of workloads is better than Δ_w. This particularly occurs in powerful CPUs (*i.e.*, modules with higher processing capacity) that require small amount of local memory to satisfy slowdown constraints. In these situations, it is possible to place workload w on a powerful CPU to provide a slowdown less than Δ_w in exchange of consuming more energy. Constraints (5b), (5c) and (5d) in Problem 2 show the updated versions of Constrains (4d), (4g) and (4i), respectively, where variable x_w is substituted with the computed value $x_{w,c}$. Constraint (5b) ensures that the sum of local memory usage of workloads assigned to the same module respects the local memory capacity of that module. Constraint (5c) enforces the capacity constraint for the remote memory in the DDC. Since $x_{w,c}$ is a constant value, variable $y_{w,c}$ is used in Constraint (5c) to compute the remote memory of workloads based on their assigned CPU modules. Constraint (5d) computes workload slowdowns. Notice that $x_{w,c}$ is a constant in this constraint, and the only variable is $y_{w,c}$.

B. Workload Placement

Algorithm 1 presents the pseudocode of IRoP that employs the deterministic rounding framework to solve F-DWLP. The algorithm IRoP gets an instance of the problem, denoted by $\mathcal{M}$ and starts its procedure with relaxing it by removing the integrality constraints in line 1. As a result, it obtains a linear program (LP), denoted by $\widetilde{\mathcal{M}}$, that can be efficiently solved using interior point methods. After solving $\widetilde{\mathcal{M}}$, IRoP obtains fractional values for $y_{w,c}$ and z_c that respect all constraints except for the integrality constraints. We denote the fractional value of these decision variables with $\widetilde{y}_{w,c}$ and $\widetilde{z}_c$, respectively. To obtain a feasible solution for $\mathcal{M}$, IRoP rounds the fractional values to either zero or one in a manner that maintains the feasibility of constraints without increasing the objective value by a significant amount. We focus on the decision variables $y_{w,c}$ in the following discussions, since the value of z_c is directly computed from the value of decision variable $y_{w,c}$. Specifically, if the value of $y_{w,c}$ is rounded to one for workload w and module c, then the value of z_c for the corresponding module must be rounded to one.

The algorithm performs the rounding of variables by considering the workloads in an iterative manner. In the iteration for workload w, the algorithm examines modules one-by-one in an order that is based on the value of decision variables $y_{w,c}$. Specifically, IRoP finds the module with the maximum value of $y_{w,c}$, denoted by c', as follows:

$$c' \leftarrow \arg\max_{c \in \mathcal{C}} \widetilde{y}_{w,c} \,. \tag{7}$$

Then, it fixes the value of decision variable $y_{w,c'}$ to one and solves the problem one more time, by starting from the current values of other decision variables. Since the change in the value of decision variables is small, the linear program solver finds the next solution very quickly or reports that the problem has become infeasible.

If $\widetilde{\mathcal{M}}$ remains feasible, the allocation of module c' is kept and the algorithm terminates the current iteration to handle the next workload in the next iteration (see lines 9 and 10). However, if the problem becomes infeasible, IRoP concludes that it is impossible to allocate module c' to workload w. Consequently, IRoP reverts the change to variable $y_{w,c}$ and solves $\widetilde{\mathcal{M}}$ again to select another module. To implement this process and ensure a module is not selected repeatedly, IRoP defines a set $\overline{\mathcal{C}}_w$ that is empty in the beginning of the iteration. Then, each time $\widetilde{\mathcal{M}}$ becomes infeasible, IRoP fixes the value of decision variable $y_{w,c'}$ to zero and adds c' to set $\overline{\mathcal{C}}_w$. If the size of $\overline{\mathcal{C}}_w$ becomes equal to $|\mathcal{C}|$, it is impossible to allocate any CPU module to workload w and the algorithm fails. In this situation, we can eliminate lower-priority workloads and repeat the process with fewer workloads.

C. Algorithm Analysis

In this section, we compute the theoretical approximation ratio and the algorithmic complexity of IRoP. Note that the slowdown of each workload w is bounded by Δ_w in F-DWLP.

Theorem 1. *Algorithm IRoP attains the approximation ratio*

$$\psi = \max\{C \cdot \widehat{N} \cdot \widehat{S}, \widehat{D}\}, \tag{8}$$

where $\widehat{N} = \max_{\substack{c \in \mathcal{C} \\ w \in \mathcal{W}}} \{N_c/\nu_w\}$ *is the maximum ratio between the number of cores and demands,* $\widehat{S} = \max_{c,c' \in \mathcal{C}}\{I^s_c/I^s_{c'}\}$ *is the maximum ratio between static energy coefficients, and* $\widehat{D} = \max_{c,c' \in \mathcal{C}}\{I^d_c/I^d_{c'}\}$ *is the maximum ratio between dynamic energy coefficients among any two modules in the DDC.*

Proof. The energy consumption term in the objective is:

$$\sum_{c \in \mathcal{C}} I^s_c \cdot z_c + I^d_c \cdot u_c \,. \tag{9}$$

TABLE V: CPU Modules Used in Evaluations.

Model	# Cores	Processing Capacity	TDP
AMD EPYC 9654	192	2	2.2
AMD EPYC 7763	128	1.5	1.7
Intel Xeon E7-8890	48	1	1

In the worst-case scenario, the value of each z_c can increase by a factor of $\frac{C \times N_c}{\widetilde{N}_w}$, as the value of $\widetilde{N}_c$ can increase by a factor of C and $\widetilde{z}_c$ can be as small as $N_w/\widetilde{N}_c$ to be larger than the left-hand side of the constraint, but it does not have to increase the value of z_c to exactly one. Therefore, the first term in (9) (*i.e.*, the static energy consumption of powered on modules) can increase by a factor of $\frac{C \times N_c}{\widetilde{N}_w}$ compared to its value in the solution of the LP. We should also note that the value of static energy consumption is not identical for all modules. Consequently, during rounding, a module with the highest static energy consumption might be selected among modules with positive value of z_c. As a result, the static energy consumption increases by at most a factor of $\widehat{S}$. To characterize the increase of the second term, we note that if the load due to each workload in one of the modules (*i.e.*, module c' that was selected in (7)) increases by a factor of α, its combined load in other modules (*i.e.*, $c \neq c'$) will decrease by a factor of $\frac{1}{\alpha}$. Therefore, the overall dynamic load characterized by the second term in (9) does not change. The only difference stems from the variations between dynamic energy consumption coefficients. Similar to the static energy consumption case, the dynamic energy consumption increases at most by a factor of $\widehat{D}$. Let $O_{\mathcal{M}}$ and $O_{\widetilde{\mathcal{M}}}$ denote the energy consumption of the optimal solution of $\mathcal{M}$ and $\widetilde{\mathcal{M}}$ used in IRoP. Also, define O_{IRoP} to be the energy consumption of the solution obtained by IRoP after rounding the decision variables of $\widetilde{\mathcal{M}}$. We have:

$$O_{\text{IRoP}} \leqslant \sum_{c \in \mathcal{C}} \left\{ \frac{C \times N_c}{\widetilde{N}_c} \cdot \widehat{S} \cdot I_c^s \cdot \widetilde{z}_c + \widehat{D} \cdot I_c^d \cdot u_c \right\} \quad (10)$$

$$\leqslant \max\left\{C \cdot \hat{N} \cdot \widehat{S}, \widehat{D}\right\} \times \sum_{c \in \mathcal{C}} \left\{ I_c^s \cdot \widetilde{z}_c + I_c^d \cdot u_c \right\} \quad (11)$$

$$= \psi \times O_{\widetilde{\mathcal{M}}} \leqslant \psi \times O_{\mathcal{M}}, \quad (12)$$

which, establishes the theorem. ∎

Theorem 2. *IRoP runs in polynomial time.*

Proof. The rounding procedure consists of calling the linear program solver at most $n \times m$ times, where n and m represent the number of workloads and CPU modules, respectively. In [24], authors prove that the complexity of solving a linear program with $n \times m$ variables using the interior point method is $O((n \times m)^{3.5})$. Therefore, in the worst-case scenario, the complexity of IRoP is $O((n \times m)^{4.5})$. ∎

V. Evaluations

A. Methodology

We conduct simulations on a large-scale DDC environment based on Google's Aquila, which supports up to 1152 hosts in one cluster [3]. RAM and CPU requirements of workloads are set based on [25], which analyzed various popular workloads used in datacenters. The simulated datacenter includes three types of processors, as described in Table V. Our implementation of the IRoP and other baselines involves approximately 1500 lines of code in Python version 3.11. The parameters of the datacenter and workload requirements are summarized in Table VI. For each of the datacenter and workload configuration, we run the simulations of different algorithms and collect the current state of the DDC (*i.e.*, available local memory and utilization of each powered on CPU) as well as the completion time of all workloads. Results are based on the average of 100 randomized configurations and we normalize the averaged results by dividing all values by the maximum.

TABLE VI: Datacenter and Workload Parameters.

Datacenter Parameters		Workload Parameters	
Symbol	Value/Range	Symbol	Value/Range
$\mathcal{C}$	1200	$\mathcal{W}$	150
F_c	See Table V	ν_w	1 - 48
N_c	See Table V	ϕ_w	1 - 2 (Normalized)
L_c	64 GB (Latency: 20ns)	μ_w	4 - 64 GB
I_c^d	25% of TDP in Table V	$P_w^m(.)$	See Equation (2)
I_c^s	75% of TDP in Table V	$P_w^f(.)$	See Equation (3)
M	20 TB (Latency: 5μs)	Δ_w	5% - 25%

In addition to IRoP, we have also implemented the following baseline algorithms for comparison:

- OPT: The optimal solution for DWLP, defined in Algorithm 1. It is obtained using the Gurobi [26] optimizer.
- HEEP [2]: A greedy algorithm which aims to minimize total energy consumption by sorting workloads and resources based on their demand and energy efficiency, respectively.
- CFM [12]: This algorithm optimizes the workload slowdown by controlling the local and remote memory ratios. It minimizes the sum of local memory-time products for each workload with the objective of optimizing the makespan.
- First-Fit: Assigns a workload to the first CPU module with enough local memory and assigns remote memory if no local memory is available.

B. Results and Discussion

Slowdown. To compare the slowdown under different algorithms, we first note that IRoP has an extra degree of freedom, namely Δ_w, compared to other algorithms. Thus, we report the normalized slowdown results of IRoP for different values of Δ_w from the set $\{0.5, 0.10, 0.15, 0.20, 0.25\}$. In Fig. 4, we report the normalized slowdown while increasing the value of η from zero to one. Note that η controls the trade-off between energy consumption and slowdown. We can see that as η increases, the normalized slowdown also increases, which is not unexpected, as higher values of η give more weight to energy consumption. These parameters allow IRoP to achieve different results based on the priorities of the datacenter operator. Fig. 4 also includes the slowdown results of CFM for comparison. We exclude reporting the slowdown of other algorithms since they do not optimize it, and their results are noticeably inferior to CFM and IRoP. While we can see IRoP outperforms CFM for

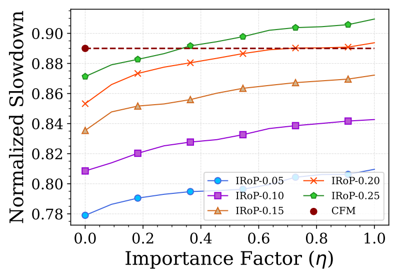


Fig. 4: Total slowdowns of IRoP for different values of slowdown threshold compared to CFM.

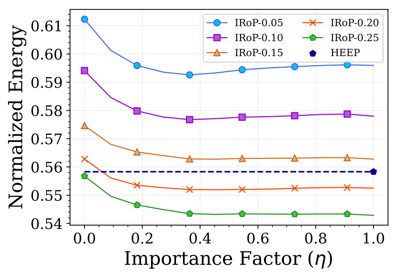


Fig. 5: Energy consumption of IRoP for different values of slowdown threshold compared to HEEP.

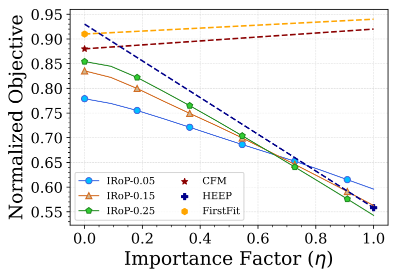


Fig. 6: Weighted sum of slowdown and energy of IRoP for different values of slowdown threshold compared to baselines.

various values of Δ_w, CFM exclusively prioritizes slowdown, making the results comparable only for $\eta = 0$. However, for other values of η, the result of CFM is represented by a dashed line. Fig. 4 shows that IRoP generally outperforms CFM, except for $\Delta_w = 0.25$ and $\eta > 0.3$ where IRoP exhibits a higher slowdown. This result is not unexpected as a loose bound on the slowdown of each workload and non-negligible importance of energy consumption lead IRoP to sacrifice the slowdown to achieve better energy consumption. Overall, over 100 randomized experiments when $\eta = 0$, the maximum, average, and minimum slowdown reduction achieved by IRoP compared to CFM are 12%, 5%, and 2%, respectively.

Energy Consumption. Fig. 5 presents the results obtained for normalized energy consumption. We can see that as Δ_w increases from 0.05 to 0.25, the energy consumption of IRoP decreases considerably, which is expected since the slowdown requirement becomes more relaxed. We have also reported the normalized energy consumption of HEEP, which is focused on energy optimization. We omit other algorithms since they do not consider energy consumption. The exclusive concentration of HEEP on the energy consumption makes it equivalent to the results of IRoP for $\eta = 1$. Nevertheless, we extended the HEEP's data point with a dashed line for further comparison. We can see that IRoP outperforms HEEP for values of Δ_w greater than 0.20. It is noteworthy that the overall slowdown of each workload in HEEP is about 0.25. Thus, we can see that for $\Delta_w = 0.2$ and $\eta \geqslant 0.2$, IRoP always provides better slowdown and lower energy consumption. Specifically, the maximum, average and minimum energy reduction achieved by IRoP over HEEP are 8%, 2.5%, and 1%, respectively.

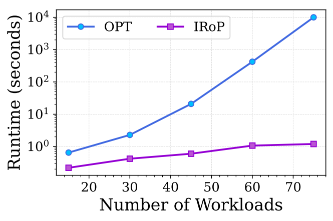
Objective. Fig. 6 presents the results obtained for the normalized value of the objective in (4a). Recall that the goal is to minimize the objective. Since other baselines can not produce different solutions for different values of Δ_w and η, we run them once and then compute the objective value based on these values and the actual slowdown and energy consumption of those solutions. Therefore, the behavior of baselines in the figure is linear. However, IRoP computes a new solution for each value in an adaptive manner to better match the need of the DDC operator. We can see that IRoP always provides a lower objective value compared to FirstFit and CFM. Furthermore, compared to HEEP, IRoP offers a lower objective value in the majority of situations. However, when $\eta \geqslant 0.7$ and $\Delta_w = 0.05$, which implies a very strict slowdown threshold for IRoP, the objective value of HEEP becomes lower. Nevertheless, as can be seen, IRoP offers a high degree of control and flexibility, which is not possible with other algorithms.

Runtime. To test the scalability of IRoP, we plotted its runtime as the number of workloads and CPU modules increase in Figs 7(a) and 7(b), respectively. The figures also show the runtime of the OPT. We observe that IRoP exhibits a mild upward trend, while OPT grows very quickly (note that the y-axis of Fig. 7(a) is logarithmically scaled). In Fig. 7(a), the number of CPU modules is fixed at 30, as increasing the number of CPU modules beyond this value results in a significantly intolerable runtime of OPT. Similarly, the number of workloads is fixed to 30 in Fig. 7(b). In these small-scale tests, IRoP introduces about 5% more slowdown and increases the energy consumption by about 7% compared to OPT, which are significantly below the theoretical bound computed in Seciton IV-C. Also, for large-scale tests, runtime of IRoP maintains under 30 seconds which is satisfactory given the testing environment.

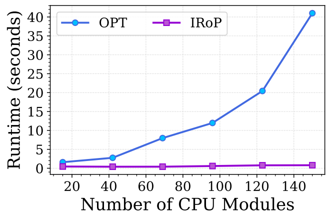
VI. Related Works

In this section, we review the existing literature on minimizing energy consumption and workload slowdown in DDCs.

Energy Consumption. Authors in [13] propose an MILP model to optimize energy consumption, considering three types of workloads: memory-intensive, IO-intensive, and processor-intensive. EERPVMM-DS [14] is a scalable heuristic solution that follows a greedy approach to minimize energy consumption while assumes a time-slotted model for the arrival of workloads submitted as virtual machines. EERP-DSCF [15] extends the work in [14] and drops the time-slotted assumption. In [2], a complete MILP model for energy consumption of DDC resources is presented to minimize the energy consumption in disaggregated architectures. Then, authors propose a heuristic, called HEEP, that uses a greedy approach based on sorting workloads and CPUs by their resource demands and energy efficiency, respectively. Authors in [17] propose a similar energy consumption model



(a) Impact of the number of workloads on the runtime of IRoP and OPT. The number of CPU modules is fixed at 30.



(b) Impact of number of CPU modules on the runtime of IRoP and OPT. The number of workloads is fixed at 30.

Fig. 7: Runtime evaluation of IRoP compared to OPT.

for CPUs while an optical and electrical interconnects are utilized as backplanes in the DDC architecture. The model involves an iterative process of filtering and prioritizing CPUs using predefined weights for energy efficiency, utilization, and communication delay. Authors in [16] propose an architectural model where the CPU and memory modules share the same board and disaggregation occurs at the level of power supply, enabling the system to save energy by suspending unutilized CPUs while providing remote access to their memory. However, the mentioned works do not consider the workload slowdown, which could result in SLA violations.

Workload Slowdown. In the context of slowdown in DDCs, several approaches have been investigated. In [11], Nvidia proposes an integrated system into the operating system (OS) which efficiently optimize page migrations between different memory levels, minimizing workload slowdown. Authors in [12], focuse on examining the impact of the remote memory to local memory ratio in such environments. The authors develop a polynomial model and utilize a latency-based workload placement, called CFM, to allocate an appropriate amount of local memory, aiming to meet the slowdown requirements. Architectural proposals like Aquila [3] propose an experimental datacenter network fabric prioritizing ultra-low latency. It incorporates the GNet protocol and custom ASIC with low-latency Remote Memory Access (RMA), achieving a sub-10 μs execution time. Additionally, HoPP [20], similar to Fastswap [12] and LegoOs [27], suggests a revised OS that decouples address capture from page faults by recording all memory access logs in the memory controller. The mentioned works explore different approaches to use in DDCs, ultimately minimizing workload slowdown while they ignore the energy consumption, which has financial and environmental impacts.

VII. Conclusion

We proposed IRoP, a workload placement algorithm minimizing DDC energy consumption while respecting a bounded slowdown for each workload. IRoP incorporates processing capacity, memory disaggregation (*i.e.*, local and remote memory), and energy efficiency models to strike a balance between energy consumption and the total slowdown of workloads. Through extensive experiments, we validated the usage of piece-wise linear functions for modeling the relationship between remote memory allocation and workload slowdown in an emulated environment. The results highlight the superior performance and flexibility of IRoP in terms of slowdown and energy consumption, effectively balancing the mentioned dual objectives based on the configurations set by datacenter owner and contributing to the workload placement problem in DDC environments (*i.e.*, DWLP). Adding an agent to monitor the slowdown of workloads and perform re-allocation is an interesting extension of our work for volatile environments.

References

[1] R. Lin, Y. Cheng *et al.*, “Disaggregated data centers: Challenges and trade-offs,” *IEEE Communications Magazine*, vol. 58, no. 2, 2020.
[2] O. O. Ajibola, T. El-Gorashi *et al.*, “Energy efficient placement of workloads in composable data center networks,” *Journal of Lightwave Technology*, vol. 39, no. 10, 2021.
[3] D. Gibson, H. Hariharan *et al.*, “Aquila: A unified, low-latency fabric for datacenter networks,” in *Proc. USENIX NSDI*, 2022.
[4] S. Ibanez, A. Mallery *et al.*, “The nanopu: A nanosecond network stack for datacenters,” in *Proc. USENIX OSDI*, 2021.
[5] P. X. Gao, A. Narayan *et al.*, “Network requirements for resource disaggregation,” in *Proc. USENIX OSDI*, 2016.
[6] Tensorflow, “Tensorflow benchmarks: A benchmark framework for tensorflow.” [Online]. Available: https://github.com/tensorflow/benchmarks/
[7] Nico, “Mini-cluster part iv : Word count benchmark,” 2015. [Online]. Available: https://tinyurl.com/2t6zftx5/
[8] J. Kim, W. Choe *et al.*, “Exploring the design space of page management for Multi-Tiered memory systems,” in *Proc. USENIX ATC*, 2021.
[9] H. A. Maruf and M. Chowdhury, “Effectively prefetching remote memory with leap,” in *Proc. USENIX ATC*, 2020.
[10] D. Masouros, C. Pinto *et al.*, “Adrias: Interference-aware memory orchestration for disaggregated cloud infrastructures,” in *Proc. IEEE HPCA*, 2023.
[11] Z. Yan, D. Lustig *et al.*, “Nimble page management for tiered memory systems,” in *Proc. ACM ASPLOS*, 2019.
[12] E. Amaro, C. Branner-Augmon *et al.*, “Can far memory improve job throughput?” in *Proc. EuroSys*, 2020.
[13] H. M. Mohammad Ali, A. Q. Lawey *et al.*, “Energy efficient disaggregated servers for future data centers,” in *Proc. EuroSys NOC*, 2015.
[14] H. M. Mohammad Ali, A. M. Al-Salim *et al.*, “Energy efficient resource provisioning with vm migration heuristic for disaggregated server design,” in *Proc. ICTON*, 2016.
[15] H. M. Mohammad Ali, T. E. H. El-Gorashi *et al.*, “Future energy efficient data centers with disaggregated servers,” *Journal of Lightwave Technology*, vol. 35, no. 24, 2017.
[16] V. Nitu, B. Teabe *et al.*, “Welcome to zombieland: Practical and energy-efficient memory disaggregation in a datacenter,” in *Proc. EuroSys*, 2018.
[17] A. D. Papaioannou, R. Nejabati *et al.*, “The benefits of a disaggregated data centre: A resource allocation approach,” in *Proc. IEEE GLOBECOM*, 2016.
[18] H. Cambazard, D. Mehta *et al.*, “Bin packing with linear usage costs – an application to energy management in data centres,” in *Proc. PPCP*, 2013.
[19] “Hpcg benchmark.” [Online]. Available: https://hpcg-benchmark.org/
[20] H. Li, K. Liu *et al.*, “Hopp: Hardware-software co-designed page prefetching for disaggregated memory,” in *Proc. IEEE HPCA*, 2023.
[21] Q. Wang, Y. Lu *et al.*, “Sherman: A write-optimized distributed b+tree index on disaggregated memory,” 2021.
[22] H. M. Mohammad Ali, A. Q. Lawey *et al.*, “Energy efficient disaggregated servers for future data centers,” in *Proc. EuroSys NOC*, 2015.
[23] D. P. Williamson and D. B. Shmoys, *The design of approximation algorithms*. Cambridge university press, 2011.
[24] N. Karmarkar, “A new polynomial-time algorithm for linear programming,” *Combinatorica*, vol. 4, 1984.
[25] A. Glawion, “How much ram do you need? different workloads explored.” 2022. [Online]. Available: https://www.cgdirector.com/how-much-ram-do-you-need/
[26] Gurobi Optimization, LLC, “Gurobi Optimizer Reference Manual,” 2023. [Online]. Available: https://www.gurobi.com
[27] Y. Shan, Y. Huang *et al.*, “Legoos: A disseminated, distributed OS for hardware resource disaggregation,” in *Proc. USENIX OSDI*, 2018.