Rethinking and Optimizing Workload Redistribution in Large-scale Internet Data Centers

Yi Zhao[1], Liang Lv[2], Yusen Li[3], Meng Shen[1], Liehuang Zhu[1], Ke Xu[2,4]
[1]School of Cyberspace Science and Technology, Beijing Institute of Technology
[2]Department of Computer Science and Technology, Tsinghua University
[3]School of Computer Science, Nankai University
[4]Zhongguancun Laboratory
Email: zhaoyi@bit.edu.cn, lvl16@tsinghua.org.cn, liyusen@nbjl.nankai.edu.cn, shenmeng@bit.edu.cn, liehuangz@bit.edu.cn, xuke@tsinghua.edu.cn

***Abstract*—Heuristic-based workload redistribution is the most commonly adopted solution to provide enhanced service performance in large-scale Internet Data Centers (IDCs). However, statistics show that they cannot perform as well as expected in real-world IDCs. In this paper, we rethink existing solutions based on real-world trace data and pinpoint two major pitfalls: (i) Sensitive to hand-tuning parameters; (ii) Reassigning only a few workloads locally at a time. The two of them jointly limit the universal applicability of existing solutions in optimizing multiple objectives fairly. To address such issues, we propose the matching-theory-based solution for workload redistribution, namely *Themis*. It is an efficient and universal solution for large-scale IDCs, which can avoid empirical parameters in optimization and reassign several workloads globally each time. Moreover, the newly proposed *Themis* can optimize multiple objectives (*e.g.*, resource utilization balancing and communication efficiency improving) simultaneously and fairly. In addition to its own performance advantages, our proposed *Themis* is also compatible with existing methods, thus adapting to a wider range of deployment scenarios. Extensive evaluations based on the trace data from two real-world IDCs demonstrate that our proposed *Themis* outperforms multiple comparison solutions, as well as the compatibility of parameter changes (*i.e.*, stability properties in terms of parameter configuration).**

***Index Terms*—Workload Redistribution, Matching Theory, Resource Utilization, Communication Efficiency.**

I. Introduction

Workload redistribution [1], [2] has been widely adopted in large-scale Internet Data Centers (IDCs) for load balancing, power saving, *etc.* The principle of workload redistribution is to migrate workloads (*e.g.*, processes, containers, virtual machines (VMs), microservices, *etc.*) among the physical hosts, seeking better performance (*e.g.*, resource utilization and communication efficiency) and lower running costs. Existing solutions to workload redistribution fall into two categories, *i.e.*, exact methods and heuristic methods. Since workload redistribution is NP-hard [3], due to high computational complexity, exact methods have limited real-world applicability. Therefore, most large-scale IDCs in the real world adopt heuristic solutions to pursue balanced efficiency and performance [4]–[6].

Although great efforts have been devoted to the workload redistribution, the status quo of IDCs in the real world is still unsatisfactory. For example, load balancing is the main objective of many workload redistribution solutions, but the statistics from top industries demonstrate that resource utilization is still severely imbalanced in real-world IDCs [7], [8].

In this paper, we rethink and improve workload redistribution in large-scale IDCs. More specifically, we perform comprehensive case studies based on real-world IDCs to analyze the performance of representative workload redistribution solutions, *i.e.*, *Local Search (LS)* and *Large Neighborhood Search (LNS)*. And then, we have several findings. First, *LS* iteratively migrates a small number of workloads globally, causing fast convergence but unstable performance. On the contrary, *LNS* repeatedly redistributes several workloads locally with relative high computational complexity, yielding stable performance but slow convergence. In addition, whether it is *LS* or *LNS*, or other existing methods, they are sensitive to empirical parameters, restricting them to optimize multiple objectives in real-world scenarios.

To address above issues, in addition to avoiding hand-tuning parameters in optimization, we should reassign several workloads globally and efficiently each time. Particularly, reassigning workloads is essentially rebuilding the matching between workloads and hosts, which inspires us to apply matching theory [9] to solve the problem. Therefore, we propose *Themis*, a matching-theory-based approach to workload redistribution. Moreover, *Themis* is implemented and extensively evaluated based on real-world IDCs. Evaluation results demonstrate the effectiveness of *Themis*. In other word, the proposed *Themis* provides a promising way to improve the performance of workload redistribution in large-scale IDCs in the real world.

Our contribution: The key contributions we have made in this paper are summarized as follows.

- Via rethinking existing solutions and case studies, we further clarify the issues of resource utilization, communication efficiency, and sensitivity to real-world scenarios.
- Based on the matching theory, we innovatively propose an efficient and universal solution *i.e.*, *Themis*, for workload redistribution with large-scale IDCs.
- In addition to the multi-objective advantages in resource utilization and communication efficiency, *Themis* can also

979-8-3503-5012-8/24/$31.00 ©2024 IEEE

be utilized as a component of existing methods to adapt to a wider range of deployment scenarios.
- It has been demonstrate that *Themis* can not only optimize multiple objectives more egalitarian, but also be compatible with parameter changes.

II. Background

To succinctly articulate our motivations and the inadequacies of existing work, in this section, we first summarize the existing workload redistribution solutions. As illustrated in Table I, these solutions fall into two categories: **exact methods** and **heuristic methods**.

TABLE I
Categories of Existing Solutions

Category		Researches
Exact methods		[10], [11]
Heuristic methods	*LS*-based heuristics	[12], [13], [14], [15], [6], [16], [17], [18], [19], [20]
	LNS-based heuristics	[21], [22], [23]
	Hybrid-based heuristics	[24], [5], [25], [26] [27], [28]

Exact methods usually formulate the workload redistribution problem into a constraint optimization form and apply optimization techniques to solve it. However, exact methods are inefficient in real-world scenarios because of high computation complexity [6].

Regarding heuristic methods, they can be further divided into three types, *i.e.*, *LS*-based heuristics, *LNS*-based heuristics, and *Hybrid*-based heuristics[1].

LS optimizes workload distribution iteratively. In each iteration, *LS* randomly selects some workloads (typically one or two workloads) and reassigns them globally with three actions, *i.e.*, *Move*, *Swap* and *Replace*. The action *Move*(c_i, h_a, h_b) shifts workload c_i from host h_a to host h_b. The action *Swap*(c_i, c_j) exchanges the locations of two workloads c_i and c_j. The action *Replace*$(c_i, h_a, c_j, h_b, h_c)$ moves workload c_j from h_b to h_c, and then moves workload c_i from h_a to h_b.

Similar to *LS*, *LNS* also works in an iterative manner. In each iteration, *LNS* selects several hosts (typically less than ten hosts) as a sub-problem, and utilizes systematic search or optimization techniques to solve the sub-problem, *i.e.*, optimizes the workload distribution on the selected hosts.

Hybrid is usually composed of multiple components, and each component runs a *LS* algorithm or a *LNS* algorithm. These components work in series or in parallel to better optimize the objective.

III. Definition of Container Redistribution

This paper considers workload redistribution in a container-based IDC. The containerized IDC is expressed in the triad of hosts, applications, and containers, denoted by $(\mathcal{H}, \mathcal{A}, \mathcal{C})$. $\mathcal{H}$ denotes the set of hosts and $\mathcal{A}$ denotes the set of applications. Each application $a \in \mathcal{A}$ is initialized as a group of containers $\mathcal{C}_a$, and each container accommodates a component of the applications. Containers of the same application communicate with each other for data transmission and synchronization to provide service. The containers of all applications constitute the full set $\mathcal{C}$. For parallel processing and robustness purpose, an application $a \in \mathcal{A}$ may have several replicas in the IDC, thus each container $c \in \mathcal{C}_a$ has the same number of replicas accordingly. The types of resources we consider in the problem are denoted as $\mathcal{R}$. For a given host $h \in \mathcal{H}$, h_r refers to its capacity of resource $r \in \mathcal{R}$. For a given container $c \in \mathcal{C}$, c_r refers to its requirement on resource $r \in \mathcal{R}$. Besides, we utilize h^c to denote the host where container c is located.

A. Objectives

In the containerized IDC, container distribution affects the applications' performance from two aspects, *i.e.*, **resource utilization** and **communication efficiency**. On the one hand, *components of the same application are usually intensive to the same type of resource, and placing them on the* ***same host*** *can easily cause* ***imbalanced resource utilization***, which further harms the availability and throughput of the corresponding application [6]. On the other hand, *when two containers are placed on* ***different physical hosts****, the* ***communication efficiency between them is much lower*** *than when they are placed on the same host [29]*. In other words, distributing the containers on a large number of hosts would increase the application's response time and degrade the performance. As a result, both factors are important for real-world systems, and we apply container redistribution to optimize both factors.

1) Resource Utilization Balancing: There are two aspects of load balancing in the IDC [10]. From the *global* perspective, for a given type of resource $r \in \mathcal{R}$, the utilization of r should be as balanced as possible across different hosts, because an overutilized host can easily become the bottleneck, thereby reducing the overall performance of the applications. From the *local* perspective, for a given host $h \in \mathcal{H}$, the utilization of different types of resources should be as equal as possible to satisfy future resource demand, because a host with much free CPU but little free memory can hardly fulfill the resource demand of new containers, which harms the service capacity of the corresponding IDC. Thus, the ideal situation is that all hosts enjoy equal utilization for resource $r \in \mathcal{R}$, which ensures the *global* load balancing as well as *local* load balancing.

We define M_r to measure the degree of utilization imbalance of resource $r \in \mathcal{R}$ in the IDC:

$$M_r = \sum_{h \in \mathcal{H}} (U_h^r - \bar{U}_r)^2, \tag{1}$$

where U_r^h is the utilization of r on host h. $\bar{U}_r$ is the optimal utilization for r, expressed as

$$\bar{U}_r = \frac{\sum_{c \in \mathcal{C}} c_r}{\sum_{h \in \mathcal{H}} h_r}. \tag{2}$$

[1]For simplicity of description, the simplified *LS*, *LNS*, and *Hybrid* refer to the corresponding heuristic method.

We apply the sum of M_r to measure the degree of overall resource utilization imbalance in the IDC, and define the Resource Utilization Balancing cost as

$$\text{OBJ}_\text{U} = \sum_{r \in \mathcal{R}} M_r. \tag{3}$$

The **minimization of OBJ_U** brings the optimization of both *global* and *local* load balancing by evenly spreading the resource demand across the hosts.

2) Overall Communication Efficiency: We define an *intra-host communication pairs* as two containers that are from the same application and located on the same host. The number of *intra-host communication pairs* reflects the communication efficiency of the applications. More *intra-host communication pairs* indicate more communications between containers happen within physical hosts, and it guarantees the overall communication efficiency.

We apply the number of *intra-host communication pairs* in the IDC to measure the Overall Communication Efficiency:

$$\text{OBJ}_\text{E} = \sum_{a \in \mathcal{A}} \sum_{c_i, c_j \in \mathcal{C}_a} I(c_i, c_j), \tag{4}$$

where $I(c_i, c_j)$ indicates whether containers c_i and c_j are located on the same host:

$$I(c_i, c_j) = \begin{cases} 1, & \text{if } h^{c_i} = h^{c_j} \\ 0, & \text{otherwise} \end{cases}. \tag{5}$$

The **maximization of OBJ_E** improves the communication efficiency in the IDC.

B. Constraints

1) Capacity Constraint: For each host $h \in \mathcal{H}$, the amount of resource $r \in \mathcal{R}$ utilized by the containers cannot exceed its capacity h_r, *i.e.*,

$$\sum_{c \in \mathcal{C}, h^c = h} c_r \leq h_r, \forall h \in \mathcal{H}, \forall r \in \mathcal{R}, \tag{6}$$

where h^c denotes the host that container c is located on.

2) Conflict Constraint: A container cannot be placed on the same host with its replicas to ensure the robustness of the application. Thus, the *Conflict Constraint* is expressed as

$$\sum_{h \in \mathcal{H}} J(h, c) = \|d_c\|, \forall c \in \mathcal{C}, \tag{7}$$

where d_c is the set of c's replicas (including c itself) and $\|d_c\|$ denotes the number of c's replicas. $J(h, c)$ indicates whether one of c's replicas is located on host h:

$$J(h, c) = \begin{cases} 1, & \text{if } h^{c'} = h \\ 0, & \text{otherwise} \end{cases}, \forall c' \in d_c. \tag{8}$$

3) Transient Constraint: To ensure the continuity of the applications, a migrated container should not be destroyed on the source host until a new instance is created on the destination host. As a result, a migrated container utilizes the resource of both source host and destination host in redistribution. We utilize h^c and h'^c to respectively denote source host and destination host of a migrated container c. The *Transient Constraint* can be expressed as

$$\sum_{h^c = h \textbf{ or } h'^c = h} c_r \leq h_r, c \in \mathcal{C}, \forall r \in \mathcal{R}, \forall h \in \mathcal{H}. \tag{9}$$

4) Migration Constraint: An application may suffer from performance degradation when migrating its containers. In practice, the number of migrated containers should not exceed a pre-configured threshold, which is expressed as

$$\sum_{c \in \mathcal{C}} K(c) \leq \lambda \cdot \|\mathcal{C}\|, \tag{10}$$

where λ denotes the maximum ratio of containers that are allowed to be migrated, and $\|\mathcal{C}\|$ denotes the total number of containers in the IDC. $K(c)$ indicates whether the location of container c is changed after redistribution. More specifically, h_b^c and h_a^c respectively represents the location of container c before and after redistribution. If $h_b^c \neq h_a^c$, $K(c) = 1$. Otherwise, $K(c) = 0$.

C. Problem Definition

In the IDC, we optimize the resource utilization and overall communication efficiency with container redistribution. The container redistribution problem is formulated as:

$$\begin{aligned} &\underset{h^c}{\text{minimize}} \quad \text{OBJ}_\text{U}, \\ &\underset{h^c}{\text{maximize}} \quad \text{OBJ}_\text{E}, \end{aligned} \tag{11}$$

The above problem (*i.e.*, Eq. (11) with the aforementioned constraints) is a non-linear mix-integer programming problem, which is NP-hard. It explains why large-scale IDCs utilize heuristics for workload redistribution.

IV. Case Study

Via the case study, we further clarify the limitation of existing workload redistribution solutions. Among the solutions summarized in Table I, we pay more attention to *LS* and *LNS*, because they are the most commonly adopted approaches in real-world IDCs. Moreover, both of them are the basic components, and their improvements can enhance the performance of *Hybrid*-based heuristics.

We apply *LS* and *LNS* to two real-world IDCs of *Baidu* (denoted as IDC_1 and IDC_2, respectively), where *LS* and *LNS* are respectively implemented based on [6] and [23]. Both methods optimize multi-objectives by optimizing the weighted sum of the objectives:

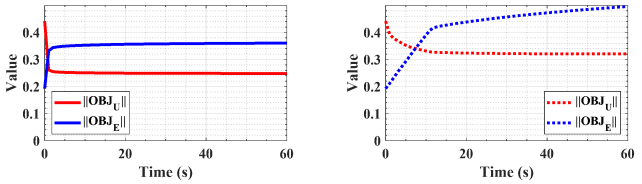
$$\textbf{minimize} \quad \text{OBJ} = \omega_u \cdot \text{OBJ}_\text{U} - \omega_e \cdot \text{OBJ}_\text{E}. \tag{12}$$

We set the weights to proper values to make sure that $\omega_u \cdot \text{OBJ}_\text{U}$ and $-\omega_e \cdot \text{OBJ}_\text{E}$ fall into similar numeric range, which is a common practice to fairly optimize multiple objectives.

In each iteration, *LS* takes an action (*i.e.*, *Move*, *Swap* or *Replace*) that improves the objective most; *LNS* selects five to ten hosts as a sub-problem and solves it with systematic search. Both algorithms are evaluated for 100 rounds. In each round, the algorithms run for 60 seconds in IDC_1 and 150

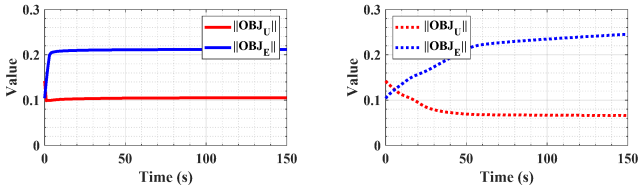
seconds in IDC_2. The maximum ratio of containers that are allowed to be reassigned is set to 20% practically.

Fig. 1 and Fig. 2 respectively illustrate the average performance of *LS* and *LNS* at each time tick as they run in the two IDCs. To understand whether these two objectives are fairly optimized, the values in the figures are after min-max normalization processing (*i.e.*, $||\text{OBJ}_\text{U}||$ and $||\text{OBJ}_\text{E}||$), where the lower bound and upper bound of each objective are respectively re-scaled to 0 and 1 in each IDC.



(a) Local Seach (b) Large Neighborhood Search

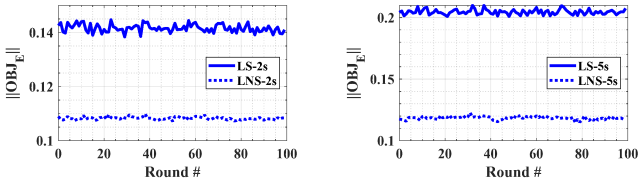
Fig. 1. Average performance in IDC_1.



(a) Local Seach (b) Large Neighborhood Search

Fig. 2. Average performance in IDC_2.

Moreover, to observe the performance fluctuation of the corresponding algorithm, Fig. 3 compares $||\text{OBJ}_\text{E}||$ in IDC_2 as *LS* and *LNS* run to the 2^{nd} second and 5^{th} second[2] in the 100 rounds of evaluation. Fig. 3 omits the curves of $||\text{OBJ}_\text{U}||$ and those in IDC_1, since the results are similar. From these figures, we have the following three findings.



(a) The 2^{nd} Second (b) The 5^{th} Second

Fig. 3. Overall communication overhead in IDC_2 at different moments.

1) Fairness Among Multiple Objectives: Although $\omega_u \cdot \text{OBJ}_\text{U}$ and $-\omega_e \cdot \text{OBJ}_\text{E}$ fall into the same numeric range in each IDC, *LNS* fails to fairly optimize OBJ_U and OBJ_E in IDC_1, while *LS* fails to fairly optimize them in IDC_2. We contribute it to that both algorithms scalar multiple objectives into a single objective by weighted sum, but it is non-trivial to assign a proper weight for each objective. Relying on hand-tuned parameters makes the algorithms inefficient to optimize multiple objectives fairly.

2) Convergence Efficiency: *LNS* converges much slower than *LS*. We attribute it to that *LNS* reassigns containers only within a few hosts in each iteration. *LNS* obtains the optimal result of the sub-problems via a relatively complex procedure (*e.g.*, systematic search or optimization tools), which however brings only a small improvement on the overall objective. *LNS* needs to solve hundreds or even thousands of sub-problems to make sure that any pair of the hosts has the chance to exchange their workloads. On the contrary, *LS* converges faster by migrating workloads efficiently in a global scope.

3) Performance Stability: As illustrated in Fig. 3, although *LS* converges faster, it performs more unstably than *LNS* in the 100 rounds of evaluations. The reason why this phenomenon occurs is that *LS* randomly reassigns a little number (typically one or two) of workloads in each iteration, making it suffer from aftereffects and resulting in unstable performance; while *LNS* co-schedules more workloads, which enables it to achieve the optimal result via multiple action sequences and results in its stable performance.

To summary, these findings indicate three important design principles of workload redistribution strategies:

- Strategy should not be sensitive to empirical parameters.
- Workloads should be efficiently migrated globally.
- The number of workloads considered in each iteration should not be too small.

Thus, if we efficiently co-schedule several workloads globally each time and avoid hand-tuning parameters, we can reach a trade-off between efficiency and stable performance.

V. Matching Theory Based Workload Redistribution

In this section, we first introduce matching theory. And then, the feasibility and challenges of applying matching theory to workload redistribution are further illustrated.

A. Matching Theory and Classic Solution

Matching theory [30] tries to build a stable matching[3] between two sides of elements. According to the number of elements on each side within a stable matching pair, matching models can be divided into three categories [9], [31], *i.e.*, *one-to-one*, *many-to-one*, and *many-to-many*. Among them, *many-to-one* is the most similar to the workload redistribution scenario. Meanwhile, College Admission Problem (CAP) is a representative model of *many-to-one* matching. In CAP, students apply for colleges, and each college accepts a specific number (*i.e.*, *quota*) of students. Each student have a strict order of preference over the colleges, and vice versa. CAP aims to build a stable matching between the students and colleges, so that each student is accepted by up to one college and no college admits more students than its quota. Two sides' mutual well-beings are jointly optimized by building such a stable matching.

Deferred Acceptance (DA) algorithm [9] is a classic solution to CAP. DA works in an iterative manner. In each iteration, every unassigned student proposes to his/her favorite college that has not rejected him/her yet. After that, each college puts

[2]Note that LS tends to converge at the 5^{th} second.

[3]The matching is stable if there do not exist two unmatched elements that prefer each other to their current partners.

its favorite *quota* proposals that it ever receives into its waitlist, and rejects all other proposals if it receives more proposals than its quota. The algorithm repeats until every student is either in the waitlist of a college, or rejected by all the colleges. The matching built by DA is stable and optimal [9].

B. Workload Redistribution with Matching Theory Perspective

Consider the process of workload redistribution: first, a set of workloads are selected and unassigned from their current locations; after that, the selected workloads are assigned to proper hosts in the IDC. Note that each workload is assigned to exactly one host and one host holds multiple workloads. In other words, there essentially exists a *many-to-one* matching between workloads and hosts, and reassigning the workloads is essentially updating this matching. These observations motivate us to apply matching theory to workload redistribution.

However, there are differences between CAP and workload redistribution. There are the following challenges in applying matching theory to the workload redistribution scenario.

1) Workload Selection: Matching theory aims to build a matching between two given sets of elements. In our problem, we have to answer two questions before performing matching. *First*, what kind of workloads should be selected and unassigned in each iteration? It is a common practice to select workloads based on the status of the hosts, and the workloads located on high-cost hosts are given higher priority. This design optimizes the objective greedily. Note that existing methods usually apply weighted summation to optimize multiple objectives, which simplifies the calculation of hosts' cost. We attempt to avoid empirical parameters in optimization, making it non-trivial to tell a host's cost. *Second*, how many workloads should be reassigned in each iteration? Reassigning more workloads helps to mitigate harmful aftereffects. However, unassigning a large number of workloads will frozen much resource because of *Transient Constraint* defined in Eq. (9), which leads to insufficient free resource to subsequent migrations. Besides, it also easily violates the *Migration Constraint* defined in Eq. (10).

2) Preference Computation: In CAP, students and colleges have their preference for the other side. However, the preferences of workloads and hosts in our problem are not as intuitive as that in CAP. In fact, we can mimic variable assignment strategies by designing proper preference schemes. For example, if the hosts prefer workloads with higher resource demand and the workloads prefer hosts with more free resources, we essentially assign workloads according to the *Worst Fit Decreasing (WFD)* strategy [32]. In other words, we need to design the preference computation mechanism carefully to better optimize multiple objectives.

3) Applicability of Classic DA Algorithm: One limitation of classic DA algorithm is its high computation complexity. In our problem, we need to perform a matching process in each iteration. Running classical DA once per iteration harms the efficiency of the solution. Moreover, compared with CAP, we have to obey the workload-redistribution-specific constraints when updating the matching. Hence, we cannot simply apply classic DA algorithm to assign selected workloads to hosts.

VI. *Themis*: Workflow and Key Mechanisms

In this section, we mitigate the challenges in Section V-B and propose a matching-theory-based workload redistribution solution, *i.e.*, *Themis*.

A. Workflow

As illustrated in Fig. 4, *Themis* is implemented as a component of the cluster orchestrator that manages the virtualized IDC. Specifically, after container migration (including new container creation) occurs, the monitor monitors various indicators of the data center in real time, including the following:

- **Host configuration:** It describes the host configuration in the IDC, including details such as ID, location, IP address, and amount of resources, and is stored in a real-time database. *Themis* reads this information from the database on demand when specifying a container redistribution solution.
- **Application information:** It describes the characteristics of distributed applications in IDC, including ID, resource requirements, number of containers and replicas owned, *etc.* This information is also stored in the real-time database as input to *Themis*.
- **Current status:** It mainly includes information such as the location of each container, the resource utilization of each host, and network bandwidth consumption. *Themis* can request this information from the monitor on demand.

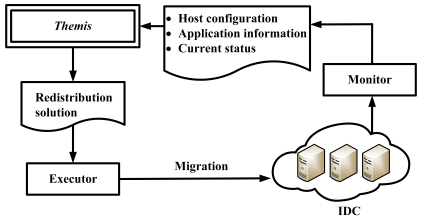


Fig. 4. Control flow of the cluster orchestrator integrated with *Themis*.

Particularly, our proposed *Themis* is triggered when the resource utilization imbalance or communication overhead exceeds the pre-configured thresholds. As illustrated in Algorithm 1, *Themis* redistributes the workloads iteratively until reaching the maximum running time. In each iteration, *Themis* first selects a set of workloads that to be reassigned (Line 4); After that, the selected workloads and *all* hosts in the IDC compute their preference lists (Line 5 to Line 8); Finally, *Themis* resorts to matching theory to assign the selected workloads to the hosts (Line 9). The mechanisms of workload selection, preference computation, and efficient matching are detailed as follows.

B. Adaptive Workload Selection

Similar to prior studies, *Themis* selects containers based on the status of the hosts. Generally, overutilized hosts account for a small proportion in the IDC, and the containers on these hosts should be given more emphasis to balance resource

Algorithm 1: Workflow of *Themis*

```
1  time_b ← time();
2  time_c ← time_b;
3  while time_c − time_b < T do
4      C_reassign ← WorkloadSelection();
5      foreach c ∈ C_reassign do
6          PreferenceComputation(c, H);
7      foreach h ∈ H do
8          PreferenceComputation(h, C_reassign);
9      EfficientMatching();
10     time_c ← time();
```

utilization. For those non-overutilized hosts, which account for a major part, we select the containers on them with a relatively low probability to optimize overall communication efficiency.

Algorithm 2: WorkloadSelection()

```
1  C_reassign ← ∅;
2  p_min ← 0;
3  p_max ← 1;
4  U ← max{U_r^h | ∀h ∈ H, ∀r ∈ R};
5  for i ← 1 to N − 1 do
6      if U ∈ [U^i, U^{i+1}) then
7          p_min ← p_min^i;
8          p_max ← p_max^i;
9          break;
10 foreach h ∈ H do
11     U^h = max{U_r^h | ∀r ∈ R};
12     if U^h > Û then
13         p(h) ← p_min + (p_max − p_min) · (U^h − Û)/(1 − Û);
14     else
15         p(h) ← p̂;
16     C_h ← Sample containers on host h with probability p(h);
17     C_reassign ← C_reassign ∪ C_h;
18 return C_reassign;
```

As illustrated in Algorithm 2, we propose an adaptive workload selection mechanism. The unbalanced degree of resource utilization is divided into N levels. According to the current unbalanced degree, *Themis* decides the numeric range of sampling probability (*i.e.*, $[p_{\min}, p_{\max}]$) for the resource-constrained hosts (Line 5 to Line 9). For a host h, if its *dominant utilization*[4] U^h is above the borderline $\hat{U}$, *Themis* sets $p(h)$ to a value that is proportional to U^h to balance resource utilization (Line 13); Otherwise, *Themis* set $p(h)$ to a borderline value $\hat{p}$ to improve global communication efficiency (Line 15). Finally, *Themis* samples the containers on h with a probability of $p(h)$. Particularly, if the number of reassigned containers has reached the threshold $\lambda \cdot \|\mathcal{C}\|$ in Eq. (10), *Themis* only selects the containers that have already been migrated.

C. Preference Computation

We design the following preference computation mechanism to fairly optimize OBJ_U and OBJ_E.

[4]The dominant utilization of host h is defined as the maximum utilization of multiple types of resources, *i.e.*, $U^h = \textbf{max}\{U_r^h | r \in \mathcal{R}\}$. A host is regarded as a resource-constrained host if its dominant utilization is above the borderline $\hat{U}$.

Algorithm 3: EfficientMatching()

```
1  Initialize a waitlist Y_h ← ∅ for each host h;
2  foreach c ∈ C_reassign do
3      Sort the hosts in its preference list L_c in descending order
       according to c's preference;
4      foreach h ∈ L_c do
5          if Y_h is empty then
6              if the free resource of h is sufficient for c then
7                  if c does not conflict with any container on h
                   then
8                      Record c in h's waitlist Y_h;
9  foreach h ∈ H do
10     Migrate the container in Y_h to host h;
```

1) Hosts' Preference: Hosts prefer the workloads that are good for balancing the resource utilization, which helps to enhance the service capacity of the IDC. More specifically, hosts prefer the workloads that reduce the gap between its current resource utilization and the optimal utilization.

For a host h and a container c, we define a standardized variable θ_c in proportion to the *Cosine Similarity*[5] between the vectors $\{\bar{U}_r - U_r^h | r \in \mathcal{R}\}$ and $\{c_r | r \in \mathcal{R}\}$, where $\bar{U}_r$ (*i.e.*, Eq. (2)) is the optimal utilization of resource r, U_r^h is the utilization of resource r on host h, c_r is container c's demand on resource r. The preference list of each host $h \in \mathcal{H}$ is generated based on θ_c. The larger θ_c, the more host h prefers container c. Specifically, a negative θ_c means that the resource utilization becomes more unbalanced if assign c to h.

2) Containers' Preference: Containers prefer the hosts that guarantee the application's performance. Particularly, containers evaluate the hosts from two aspects. *First*, a container prefers a host accommodating more of its *peers*[6], which is expected to improve the communication efficiency of the application. *Second*, a container likes a host whose free resource is homogeneous to its resource demand, which ensures that the container's *dominant demand*[7] will be better satisfied.

For a container c and a host h, we define two standardized variables θ_p^c and θ_h^u. θ_p^c is in proportional to the number of c's peers on h. θ_h^u is in proportion to the *Cosine Similarity* between the vectors $\{c_r | r \in \mathcal{R}\}$ and $\{\frac{\bar{U}_r - U_r^h}{|\bar{U}_r - U_r^h|} | r \in \mathcal{R}\}$, where c_r is the container c's demand on resource r and $\frac{\bar{U}_r - U_r^h}{|\bar{U}_r - U_r^h|}$ is the normalized gap between h's resource utilization and the optimal utilization. The preference list of container c is generated based on $\theta_h = \theta_h^c + \theta_h^u$. The larger θ_h, the more container c likes host h.

D. Efficient Matching for Workload Redistribution

We design an efficient matching mechanism in Algorithm 3 to assign the selected containers to the hosts. Particularly, each

[5]The *Cosine Similarity* of two non-zero vectors is defined as **similarity** $= (A \cdot B)/(|A| \cdot |B|)$. The output ranges from -1 to 1, where -1 means non-similar and 1 means totally similar.

[6]The peers of container c refer to the containers that belong to the same application with c. These containers will communicate with c in operation.

[7]The dominant demand of container c is defined as its maximum demand on different types of resources, *i.e.*, $\textbf{max}\{c_r | r \in \mathcal{R}\}$.

selected container c proposes for at most one host (denoted as h) that satisfies the following two conditions. First, the waitlist of h is empty (Line 5). Second, migrating c to h will not violate any constraint (Line 6 and Line 7). In this way, the containers can be assigned to the hosts efficiently while obeying the constraints defined in Section III-B.

VII. Evaluation

In the evaluation, we seek to understand: 1) What is the overall performance of *Themis*? 2) Is *Themis* sensitive to parameter settings? 3) Can *Themis* improve the performance of hybrid solutions? Accordingly, the relevant results are summarized as follows.

- **Overall performance:** *Themis* outperforms representative workload redistribution solutions in terms of both resource utilization balancing and overall communication efficiency. Particularly, *Themis* can optimize multiple objectives more egalitarian than existing solutions.
- **Sensitivity:** By halving and doubling the value of parameters in *Themis*, it has been demonstrated that *Themis* works stable and is insensitive to parameter settings.
- **Performance improvement to *Hybrid*-based Heuristics:** We use *Themis* to replace the *LS* components and *LNS* components of a hybrid solution, and find that the updated solution performs much better than the original design. *Themis* shows the potential to improve the performance of *Hybrid*-based heuristics.

A. Experimental Configuration

1) Environment: We perform the evaluations in two real-world IDCs of *Baidu* (respectively denoted as IDC_1 and IDC_2). IDC_1 contains $2,500+$ hosts and $12,000+$ containers from 25 applications. IDC_2 contains $5,900+$ hosts and $21,000+$ containers from 36 applications. In the IDCs, we apply workload redistribution to improve the overall communication efficiency and balance the usage of three types of resources, namely CPU, memory (MEM) and storage (SSD).

2) Solutions: The solutions that incorporate our proposed *Themis* and the selected baseline methods for comparison are listed as follows:

- *Themis:* Our proposed matching-theory-based approach.
- *Noisy Local Search (NLS) [6]*: It is the winner solution to Google Machine Reassignment Problem (GMRP). And *NLS* optimizes the distribution of the workloads by actions *Move*, *Swap*, and *Replace*.
- *Large Neighborhood Search (LNS) [23]*: In terms of workload redistribution, *LNS* samples several high-cost hosts as sub-problems, and utilizes systematic search to redistribute workloads on the sampled hosts.
- *Sweep&Search (S&S) [33]*: It is a *Hybrid*-based heuristic that redistributes workloads in two stages. In the first stage, *S&S* migrates workloads from the underutilized hosts to the medium utilized hosts, and then migrates workloads from the overutilized hosts to the spared underutilized ones. In the second stage, it applies local search to further improve the workload distribution. The first stage is essentially a *LNS* component, and the second stage is essentially a *LS* component.
- *Worst Fit Decreasing (WFD)*: In addition to the matching theory, multi-resource bin packing strategy can also assign the selected workloads to the hosts [32], [34]. To illustrate the advantages of matching theory, we replace the matching process in *Themis* (*i.e.*, Line 9 in Algorithm 1) with *WFD* strategy. *WFD* is a bin packing strategy that focuses on load balancing. To assign a set of workloads to the hosts, *WFD* sorts the workloads by their resource demand in descending order, and in turn places them to the host with the most free resource.

3) Parameter Setting: Regarding with each IDC, we evaluate every solution for 100 rounds. In each round, the solutions run for 60 seconds in IDC_1, while run for 150 seconds in IDC_2. Particularly, if the number of migrated containers reaches the threshold defined in Eq. (10), the solutions only reassign the containers that have already been migrated.

Regarding the parameter configuration of our proposed *Themis*, $p^i_{\textbf{min}} \in [0.002, 0.01]$, $p^i_{\textbf{max}} \in [0.01, 0.02]$, and $\hat{p} = 0.001$. Moreover, evaluations demonstrate that *Themis* is insensitive to these values.

In each iteration, *LNS* sorts the hosts by their contribution to OBJ in Eq. (12). Among them, the top 4% are regarded as high-cost hosts, and five to ten high-cost hosts are randomly chosen as a sub-problem.

For a host $h \in \mathcal{H}$, *S&S* uses $\sum_{r\in\mathcal{R}}(1 - U^h_r)$ to measure the free resource of h. The hosts are sorted by their free resource in ascending order. Then the top 2% and tail 2% are respectively regarded as underutilized and overutilized hosts, and the remaining are regarded as medium utilized hosts.

NLS, *LNS*, and *SS* optimize multiple objectives by scaling them into a single one with weighted sum as in Eq. (12). The weights are set to make $\omega_u \cdot \text{OBJ}_\text{U}$ and $-\omega_e \cdot \text{OBJ}_\text{E}$ fall into similar numerical ranges and optimized fairly.

In terms of *WFD*, the free resource of a host h is measured by $\sum_{r\in\mathcal{R}}(1 - U^h_r)$, and the resource demand of a container c is measured by $\textbf{max}\{c_r | r \in \mathcal{R}\}$.

TABLE II
Average Performance of Different Solutions

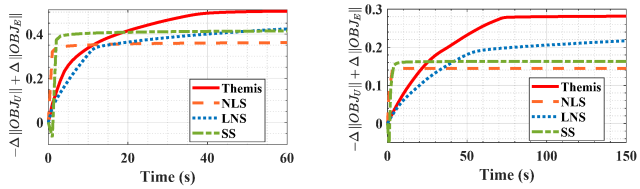
		Initial Value	*Themis*	*NLS*	*LNS*	*S&S*	*WFD*
$\textbf{IDC}_1$	OBJ_U	261.46	**163.54**	188.06	215.77	164.66	178.67
	OBJ_E	3263	6548	5508	**7316**	5389	3236
	$\|\text{OBJ}_\text{U}\|$	0.44	**0.18**	0.25	0.32	0.19	0.22
	$\|\text{OBJ}_\text{E}\|$	0.19	0.44	0.36	**0.50**	0.35	0.19
	$\Delta\|\text{OBJ}_\text{U}\|$	\	**−0.26**	−0.19	−0.08	−0.25	−0.22
	$\Delta\|\text{OBJ}_\text{E}\|$	\	0.25	0.17	**0.31**	0.16	0
$\textbf{IDC}_2$	OBJ_U	528.39	**343.41**	475.81	419.70	425.18	447.94
	OBJ_E	3515	**8570**	7089	8186	6536	4049
	$\|\text{OBJ}_\text{U}\|$	0.14	**0.01**	0.11	0.07	0.07	0.09
	$\|\text{OBJ}_\text{E}\|$	0.10	**0.26**	0.21	0.24	0.20	0.12
	$\Delta\|\text{OBJ}_\text{U}\|$	\	**−0.13**	−0.03	−0.07	−0.07	−0.05
	$\Delta\|\text{OBJ}_\text{E}\|$	\	**0.16**	0.11	0.14	0.10	0.02

B. Overall Performance

Table II summarizes the average performance of different solutions with the 100 rounds of evaluations. In IDC_1, our proposed *Themis* reduces OBJ_U by 37.5% and enhances OBJ_E by 100.7%. In IDC_2, *Themis* decreases OBJ_U by 35% and improves OBJ_E by 143.8%. These results demonstrate that *Themis* outperforms other solutions in terms of both objectives, *i.e.*, resource utilization balancing (minimization of OBJ_U) and overall communication efficiency (maximization of OBJ_E) in almost all the cases.

To observe whether OBJ_U and OBJ_E are fairly optimized, Table II further gives the value of the objectives that after min-max normalization, (*i.e.*, $\|OBJ_U\|$ and $\|OBJ_E\|$), where the lower bound and upper bound of each objective are respectively re-scaled to 0 and 1. Note that $\Delta\|OBJ_U\|$ in the table is negative, because we try to minimize OBJ_U in optimization. In IDC_1, *Themis* reduces $\|OBJ_U\|$ by 0.26 and enhances $\|OBJ_E\|$ by 0.25. In IDC_2, *Themis* lowers $\|OBJ_U\|$ by 0.13 and increases $\|OBJ_E\|$ by 0.16. Therefore, in addition to optimizing multiple objectives simultaneously, *Themis* optimizes multiple objectives more fairly than other solutions.

To understand the converge speed of the solutions, Fig. 5 demonstrates the average improvement of the objectives as the solutions run. *LNS* and *SS* converge faster than the other methods, followed by *Themis*. Particularly, *Themis* outperforms the other solutions after running for 20 seconds in the two IDCs.



(a) IDC_1 (b) IDC_2

Fig. 5. Convergence speed of different solutions.

To observe the performance when different ratio of containers that are allowed to be reassigned, Table III compares the average performance of these solutions as λ in Eq. (10) are set to 10% and 30%. For clarity, the table only gives the value that after min-max normalization, *i.e.*, $\|OBJ_U\|$ and $\|OBJ_E\|$. Since migrating more containers can optimize the objectives better, all solutions perform better as λ increases. Moreover, independent of the value of λ, our proposed *Themis* outperforms the other solutions in terms of overall improvement as well as fairness in multi-objective optimization.

C. Parameter Sensitivity of Themis

The newly proposed *Themis* requires to specify the range of sampling probability $[p^i_{\mathbf{min}}, p^i_{\mathbf{max}}]$ as well as the borderline probability $\hat{p}$ in operation. In each IDC, we halve and double $p^i_{\mathbf{min}}$, $p^i_{\mathbf{max}}$, and $\hat{p}$ to verify the sensitivity of *Themis*. As illustrated in Table IV, no matter which IDC it is, parameter changes cannot affect the performance (*i.e.*, minimization of OBJ_U and maximization of OBJ_E) of *Themis*. This reveals that our proposed *Themis* is insensitive to the parameter settings, which guarantees a stable performance across different IDCs.

TABLE III
AVERAGE PERFORMANCE UNDER DIFFERENT REASSIGNMENT THRESHOLDS

Different Ratios		Initial Value	*Themis*	*NLS*	*LNS*	*S&S*
$\mathbf{IDC_1}$	$\|OBJ_U\|$	0.44	**0.24**	0.29	0.35	**0.24**
(λ=10%)	$\|OBJ_E\|$	0.19	**0.30**	0.26	0.34	0.26
$\mathbf{IDC_1}$	$\|OBJ_U\|$	0.44	0.18	0.23	0.29	**0.16**
(λ=30%)	$\|OBJ_E\|$	0.19	0.46	0.52	**0.65**	0.51
$\mathbf{IDC_2}$	$\|OBJ_U\|$	0.14	**0.04**	0.10	0.08	0.08
(λ=10%)	$\|OBJ_E\|$	0.10	0.16	0.15	**0.17**	0.14
$\mathbf{IDC_2}$	$\|OBJ_U\|$	0.14	**0.01**	0.13	0.06	0.07
(λ=30%)	$\|OBJ_E\|$	0.10	**0.33**	0.32	0.32	0.27

TABLE IV
PARAMETER SENSITIVITY OF *Themis*

		Initial Value	*Themis* (1×)	*Themis* (0.5×)	*Themis* (2×)
$\mathbf{IDC_1}$	OBJ_U	261.46	163.54	163.82	164.38
	OBJ_E	3263	6548	6583	6419
$\mathbf{IDC_2}$	OBJ_U	528.39	343.41	342.50	346.02
	OBJ_E	3515	8570	8567	8555

D. Performance Improvement to Hybrid-based Heuristics

As a newly proposed method, if *Themis* is compatible with existing methods, it will have a wider range of deployment scenarios. To understand whether the newly proposed *Themis* can improve the performance of *Hybrid*-based heuristics, we replace the *LNS* component and *NLS* component of *S&S* [33] with *Themis*, which evolved into a new workload redistribution scheme, namely *Themis-Based S&S*. More specifically, *Themis-Based S&S* works as follows. In the first stage, it utilizes Algorithm 3 to assign the containers on the underutilized hosts to the medium utilized hosts, and to assign the containers on the overutilized hosts to the underutilized hosts. In the second stage, it applies Algorithm 1 to optimize the container distribution in the whole IDC.

TABLE V
IMPROVEMENTS IN HYBRID METHODS

		Initial Value	*S&S*	*Themis-based S&S*
$\mathbf{IDC_1}$	OBJ_U	261.46	164.66	**163.93**
	OBJ_E	3263	5389	**6626**
$\mathbf{IDC_2}$	OBJ_U	528.39	425.18	**355.48**
	OBJ_E	3515	6536	**8547**

As illustrated in Table V, *Themis-based S&S* significantly outperforms *S&S* in all IDCs. Particularly, *Themis-based S&S* optimizes OBJ_U and OBJ_E much more fairly than the original

design. Therefore, our proposed *Themis* has the potential to improve the performance of existing *Hybrid*-based heuristics. In other word, the newly proposed method has good compatibility properties. In addition to directly serving workload redistribution tasks, it can also be utilized as an auxiliary tool for existing methods to further optimize various objectives.

VIII. Related Work

With the trend of widespread application of deep learning technologies [35]–[37], data is one of the important resources, and the optimization of workload redistribution in IDCs can provide better services for intelligent technologies. In this section, we survey the solutions to the representative workload redistribution problems, such as Google Machine Reassignment Problem (GMRP), virtual machine reassignment, and container redistribution. Finally, we discuss the applications of matching theory in IDCs.

A. Existing Workload Redistribution Solutions

Exact Methods. Rui *et al.* [10] propose a multi-objective optimization solution to rebalance VM placement. Terra *et al.* [11] adopt a boolean optimization solver for VM reassignment. Although optimization-based solutions can find the optimal solution, they are inefficient in large-scale scenarios due to high complexity [6].

***LS*-based Heuristics.** Gavranović *et al.* [6] propose the winner solution to GMRP, *i.e.*, *Noisy Local Search (NLS)*, which combines *LS* and noising strategy in workload redistribution. Turky *et al.* [16], [19] propose a series of parallel algorithms, where multiple *LS* modules work together to avoid local optima. An improved great deluge algorithm is proposed in [18], which considers more neighborhoods to navigate the search space. A simulated annealing algorithm is designed by Turky *et al.* [20], which uses *Move* and *Swap* to optimize workload distribution. Guo *et al.* [12] and Li *et al.* [13] improve load balancing in cloud with particle swarm optimization based algorithms. An ant colony optimization based solution is proposed by Kaewkasi *et al.* [14], which can balance resource usages for Docker. In addition, Lin *et al.* [15] utilize the ant colony optimization technique to optimize multi-objectives in container-based cloud.

***LNS*-based Heuristics.** Mrad *et al.* [21] formulate GMRP as a Mixed-Integer Programming (MIP) problem, and utilize a Mixed Integer Linear Programming (MILP) solver to solve the sub-problems. Brandt *et al.* [22] and Mehta *et al.* [23] transform GMRP into Constrain Programming (CP) forms, and respectively apply a CP solver and systematic search technique to solve the sub-problems.

***Hybrid*-based Heuristics.** Saber *et al.* [25] propose a series of comprehensive methods that integrate *LS* algorithms and genetic algorithms. Jaskowski *et al.* [26] divide the problem into sub-problems, and utilize MIP solver to solve the sub-problems, and then apply hill climbing technique to further improve the results. Butelle *et al.* [27] propose a fast machine reassignment approach, where a *LS* component and a *LNS* component run in parallel to produce better results. Zhang *et al.* [33] design a two-step solution *Sweep&Search*, which first utilize *LNS* to roughly balance resource utilization and then applies *LS* to find-tune the container distribution.

B. Applications of Matching Theory in IDCs

There has been some explorations on the utilization of matching theory in IDCs. For example, Dhillon *et al.* [38] map VM placement into the stable-marriage problem and place the VMs according to their characteristics (*i.e.*, computation intensive, memory intensive, *etc.*). Azougaghe *et al.* [39] propose a matching game based algorithm for secure-aware VM placement. Xu *et al.* [40] design an egalitarian matching framework for VM placement. Wang *et al.* [41] apply matching theory to place VMs, thereby optimizing resource utilization as well as the quality of service. Xu *et al.* [42] propose a matching theory based resource scheduling approach to optimize the resource utilization of IDC as well as the response time of jobs. Chen *et al.* [43] adopt matching theory in container deployment to improve resource utilization rate. Sangar *et al.* [44] apply the stable marriage model to build *one-to-one* matchings between overutilized hosts and underutilized hosts, and then migrates the workloads from overutilized hosts to the underutilized ones to balance CPU utilization.

These solutions cannot be applied to the issue that this paper focuses on. *First*, most of above researches focus on workload placement (*i.e.*, deploy new workloads into the IDC), which is different from the workload redistribution issue. *Second*, the models proposed by these researches are simple and ideal, which makes them inapplicable in real-world systems.

IX. Conclusion

Workload redistribution has been widely studied in recent years. However, the performance (*e.g.*, resource utilization balancing, overall communication efficiency, fairness among multiple objectives, parameter sensitivity, and compatibility with existing methods) of existing solutions in real-world scenarios is still unsatisfactory. This paper aims to rethink existing approaches in depth and propose an efficient yet universal solution to workload redistribution with large-scale IDCs in the real world.

Based on comprehensive case studies, we find that reassigning multiple workloads globally each time and avoiding experience-based parameters are critical for effective workload redistribution. Inspired by the observations, we propose the matching-theory-based workload redistribution solution, *i.e.*, *Themis*. Extensive evaluations on real-world IDCs demonstrate that our proposed *Themis* significantly outperforms the alternative solutions and provides a promising way to improve workload redistribution in real-world scenarios.

Acknowledgment

This work in this paper was in part supported by the National Natural Science Foundation of China with Grant No. 62202258, No. 62221003, No. 62222201, No. U23A20304, and No. U22B2031, China National Funds for Distinguished Young Scientists with Grant No. 62425201 and

Beijing Outstanding Young Scientist Project with Grant No. BJJWZYJH01201910003011. Yi Zhao and Liang Lv are the co-first authors. Ke Xu is the corresponding author.

REFERENCES

[1] H. Feng, Y. Deng, Y. Zhou, and G. Min, "Towards Heat-Recirculation-Aware Virtual Machine Placement in Data Centers," *IEEE TNSM*, vol. 19, no. 1, pp. 256–270, 2022.

[2] Y. Ran, H. Hu, Y. Wen, and X. Zhou, "Optimizing Energy Efficiency for Data Center via Parameterized Deep Reinforcement Learning," *IEEE TSC*, vol. 16, no. 2, pp. 1310–1323, 2023.

[3] L. Lv, Y. Zhang, Y. Li, K. Xu, D. Wang, W. Wang, M. Li, X. Cao, and Q. Liang, "Communication-Aware Container Placement and Reassignment in Large-Scale Internet Data Centers," *IEEE JSAC*, vol. 37, no. 3, pp. 540–555, 2019.

[4] T. Saber, A. Ventresque, J. Marques-Silva, J. Thorburn, and L. Murphy, "MILP for the Multi-objective VM Reassignment Problem," in *Proceedings of IEEE ICTAI*, 2015, pp. 41–48.

[5] T. Saber, J. Marques-Silva, J. Thorburn, and A. Ventresque, "Exact and Hybrid Solutions for the Multi-Objective VM Reassignment Problem," *International Journal on Artificial Intelligence Tools*, vol. 26, no. 1, pp. 1 760 004:1–35, 2017.

[6] H. Gavranović and M. Buljubašić, "An efficient local search with noising strategy for google machine reassignment problem," *Annals of Operations Research*, pp. 1–13, 2014.

[7] H. M. Afsar, C. Artigues, E. Bourreau, and S. Kedad-Sidhoum, "Machine Reassignment Problem: The ROADEF/EURO Challenge 2012," *Annals of Operations Research*, vol. 242, no. 1, pp. 1–17, 2016.

[8] C. Lu, K. Ye, G. Xu, C.-Z. Xu, and T. Bai, "Imbalance in the Cloud: an Analysis on Alibaba Cluster Trace," in *Proceedings of IEEE Big Data*, 2017, pp. 2884–2892.

[9] D. Gale and L. S. Shapley, "College Admissions and the Stability of Marriage," *American Mathematical Monthly*, pp. 9–15, 1962.

[10] L. Rui, Q. Zheng, X. Li, and W. Jie, "A Novel Multi-objective Optimization Scheme for Rebalancing Virtual Machine Placement," in *Proceedings of IEEE CLOUD*, 2017, pp. 710–717.

[11] M. Terra-Neves, I. Lynce, and V. Manquinho, "Virtual Machine Consolidation Using Constraint-based Multi-Objective Optimization," *Journal of Heuristics*, vol. 25, no. 3, pp. 339–375, 2019.

[12] Y. Guo and W. Yao, "A Container Scheduling Strategy Based on Neighborhood Division in Micro Service," in *Proceedings of IEEE/IFIP NOMS*, 2018, pp. 1–6.

[13] J. Li, B. Liu, W. Lin, P. Li, and Q. Gao, "An Improved Container Scheduling Algorithm Based on PSO for Big Data Applications," in *International Symposium on Cyberspace Safety and Security*. Springer, 2019, pp. 516–530.

[14] C. Kaewkasi and K. Chuenmuneewong, "Improvement of Container Scheduling for Docker using Ant Colony Optimization," in *Proceedings of IEEE KST*, 2017, pp. 254–259.

[15] M. Lin, J. Xi, W. Bai, and J. Wu, "Ant Colony Algorithm for Multi-Objective Optimization of Container-Based Microservice Scheduling in Cloud," *IEEE Access*, vol. 7, pp. 83 088–83 100, 2019.

[16] A. Turky, N. R. Sabar, A. Sattar, and A. Song, "Parallel Late Acceptance Hill-Climbing Algorithm for the Google Machine Reassignment Problem," in *Australasian Joint Conference on Artificial Intelligence*. Springer, 2016, pp. 163–174.

[17] A. Turky, N. R. Sabar, and A. Song, "An Evolutionary Simulating Annealing Algorithm for Google Machine Reassignment Problem," in *Intelligent and Evolutionary Systems*, 2017, pp. 431–442.

[18] A. Turky, N. R. Sabar, A. Sattar, and A. Song, "Multi-neighbourhood Great Deluge for Google machine reassignment problem," in *Asia-Pacific Conference on Simulated Evolution and Learning*, 2017, pp. 706–715.

[19] A. Turky, N. R. Sabar, and A. Song, "Cooperative Evolutionary Heterogeneous Simulated Annealing Algorithm for Google Machine Reassignment Problem," *Genetic Programming and Evolvable Machines*, vol. 19, no. 1-2, pp. 183–210, 2018.

[20] G. M. Portal, M. Ritt, L. M. Borba, and L. S. Buriol, "Simulated Annealing for the Machine Reassignment Problem," *Annals of Operations Research*, vol. 242, no. 1, pp. 93–114, 2016.

[21] M. Mrad, A. Gharbi, M. Haouari, and M. Kharbeche, "An Optimization-based Heuristic for the Machine Reassignment Problem," *Annals of Operations Research*, vol. 242, no. 1, pp. 115–132, 2016.

[22] F. Brandt, J. Speck, and M. Völker, "Constraint-based Large Neighborhood Search for Machine Reassignment," *Annals of Operations Research*, vol. 242, no. 1, pp. 63–91, 2016.

[23] D. Mehta, B. O'Sullivan, and H. Simonis, "Comparing Solution Methods for the Machine Reassignment Problem," in *Proceedings of CP*, 2012, pp. 782–797.

[24] T. Saber, A. Ventresque, I. Brandic, J. Thorburn, and L. Murphy, "Towards a Multi-objective VM Reassignment for Large Decentralised Data Centres," in *Proceedings of IEEE/ACM UCC*, 2015, pp. 65–74.

[25] T. Saber, X. Gandibleux, M. O'Neill, L. Murphy, and A. Ventresque, "A Comparative Study of Multi-Objective Machine Reassignment Algorithms for Data Centres," *Journal of Heuristics*, pp. 1–32, 2019.

[26] W. Jaśkowski, M. Szubert, and P. Gawron, "A Hybrid MIP-based Large Neighborhood Search Heuristic for Solving the Machine Reassignment Problem," *Annals of Operations Research*, vol. 242, no. 1, pp. 33–62, 2016.

[27] F. Butelle, L. Alfandari, C. Coti, L. Finta, L. Létocart, G. Plateau, F. Roupin, A. Rozenknop, and R. W. Calvo, "Fast Machine Reassignment," *Annals of Operations Research*, vol. 242, no. 1, pp. 133–160, 2016.

[28] N. R. Sabar, A. Song, and M. Zhang, "A Variable Local Search based Memetic Algorithm for the Load Balancing Problem in Cloud Computing," in *European Conference on the Applications of Evolutionary Computation*. Springer, 2016, pp. 267–282.

[29] T. Yu, S. A. Noghabi, S. Raindel, H. Liu, J. Padhye, and V. Sekar, "FreeFlow: High Performance Container Networking," in *Proceedings of ACM HotNets*, 2016, pp. 43–49.

[30] A. E. Roth and A. Oliveira, "Two-Sided Matching: A Study in Game-Theoretic Modeling and Analysis," in *Econometric Society Monograph Series*, 1990.

[31] F. Echenique and J. Oviedo, "A Theory of Stability in Many-to-Many Matching Markets," *Theoretical Economics*, vol. 1, no. 2, pp. 233–273, 2006.

[32] M. R. Garey and D. S. Johnson, *Approximation Algorithms for Bin Packing Problems: A Survey*. Springer Vienna, 1981, pp. 49–106.

[33] Y. Zhang, Y. Li, K. Xu, D. Wang, M. Li, X. Cao, and Q. Liang, "A Communication-Aware Container Re-Distribution Approach for High Performance VNFs," in *Proceedings of IEEE ICDCS*, 2017.

[34] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, "Multi-resource Packing for Cluster Schedulers," in *Proceedings of ACM SIGCOMM*, 2014, pp. 455–466.

[35] Y. Zhao, K. Xu, J. Chen, and Q. Tan, "Collaboration-Enabled Intelligent Internet Architecture: Opportunities and Challenges," *IEEE Network*, vol. 36, no. 5, pp. 98–105, 2022.

[36] Y. Zhao, M. Qiao, H. Wang, R. Zhang, D. Wang, and K. Xu, "Friendship Inference in Mobile Social Networks: Exploiting Multi-Source Information With Two-Stage Deep Learning Framework," *IEEE/ACM ToN*, vol. 31, no. 2, pp. 542–557, 2023.

[37] G. Zhou, Q. Li, Y. Liu, Y. Zhao, Q. Tan, S. Yao, and K. Xu, "FedPAGE: Pruning Adaptively Toward Global Efficiency of Heterogeneous Federated Learning," *IEEE/ACM ToN*, 2023.

[38] J. S. Dhillon, S. Purini, and S. Kashyap, "Virtual Machine Coscheduling: A Game Theoretic Approach," in *Proceedings of IEEE/ACM UCC*, 2013, pp. 227–234.

[39] A. Azougaghe, O. A. Oualhaj, M. Hedabou, M. Belkasmi, and A. Kobbane, "Many-to-One Matching Game Towards Secure Virtual Machines Migration in Cloud Computing," in *Proceedings of IEEE ACOSIS*, 2016, pp. 1–7.

[40] H. Xu and B. Li, "Egalitarian Stable Matching for VM Migration in Cloud Computing," in *Proceedings of IEEE INFOCOM WKSHPS*, 2011, pp. 631–636.

[41] J. V. Wang, K.-Y. Fok, C.-T. Cheng, and K. T. Chi, "A Stable Matching-Based Virtual Machine Allocation Mechanism for Cloud Data Centers," in *Proceedings of IEEE SERVICES*, 2016, pp. 103–106.

[42] X. Xu, H. Yu, and X. Pei, "A Novel Resource Scheduling Approach in Container Based Clouds," in *Proceedings of IEEE CSE*, 2014, pp. 257–264.

[43] F. Chen, X. Zhou, and C. Shi, "The Container Deployment Strategy Based on Stable Matching," in *Proceedings of IEEE ICCCBDA*, 2019, pp. 215–221.

[44] D. Sangar, H. Haugerud, A. Yazidi, and K. Begnum, "A Decentralized Approach for Homogenizing Load Distribution: In Cloud Data Center Based on Stable Marriage Matching," in *Proceedings of ACM MEDES*, 2019, pp. 292–299.