Improving Data Center Efficiency: A Combinatorial Optimization Framework for Resource Allocation

Thiago Felipe Silva da Cruz[1] e Karl Heinz Kienitz[2]
[1]Instituto Tecnológico de Aeronáutica, São José dos Campos/SP - Brasil
[2]Instituto Tecnológico de Aeronáutica, São José dos Campos/SP - Brasil

***Resumo*— This work addresses resource allocation optimization in data centers, formulated as a variation of the bin packing problem. Due to the high costs of on-premises infrastructures, strategies for migrating workloads to the public cloud and shutting down physical and virtual machines are investigated to enhance operational cost efficiency. Preliminary experiments demonstrate the effectiveness of the proposed algorithm at different scales, highlighting its performance and limitations in complex scenarios. Execution time metrics and the number of simulations indicate that the algorithm is efficient for small to medium-sized tasks but faces challenges under extensive demands.**

***Palavras-Chave*— resource allocation, algorithm, datacenter.**

I. Introduction and Motivation

Optimization problems in computational theory benefit from robust mathematical frameworks and methodologies. Significant contributions by researchers such as Papadimitriou and Steiglitz have been important in advancing solutions to such challenges [1]. The need for solving combinatorial optimization problems often stems from practical applications requiring efficient resource allocation, such as logistics, scheduling, network design, and production planning. Effective resource allocation in these fields is essential for achieving desired outcomes, reducing costs, and maximizing efficiency.

This research aims to apply existing knowledge in combinatorial optimization to a specific resource allocation problem. The problem is formulated as a variation of the bin packing problem, inspired by works such as "Bin Packing Approximation Algorithms: Survey and Classification"by Coffman et al. [2].

The consolidation of emerging technologies such as cloud computing [3]–[5] introduces novel challenges like distributed computing, load balancing and real-time processing of large data sets [6]–[11]. Economically, cloud data centers present a more cost-effective option compared to on-premises data centers, which can be large and expensive to maintain [12]–[14].

The migration of on-premises workloads to the cloud can be approached from two perspectives. The first focuses on costs: how can we deactivate machines in on-premises data centers in parallel with the migration process? Deactivating machines can reduce costs. The second addresses to critical systems that cannot go offline. In this case, workloads can be redistributed across different machines, allowing older machines to be decommissioned and the data center infrastructure to be modernized. This must occur in parallel with the migration of the workload to another infrastructure, whether temporary or permanent. This research aims to contribute to the field of resource allocation in combinatorial settings by providing a framework to address challenges related to decommissioning in large data centers.

A. The Problem

This section begins by defining the problem through a characterization of a large-scale data center. This includes fundamental components and simplified constraints governing the system, along with considerations of workload migration [15]–[21]. The system consists of a data center with machines (servers) and a workload representing processes that demand capacity from these machines.

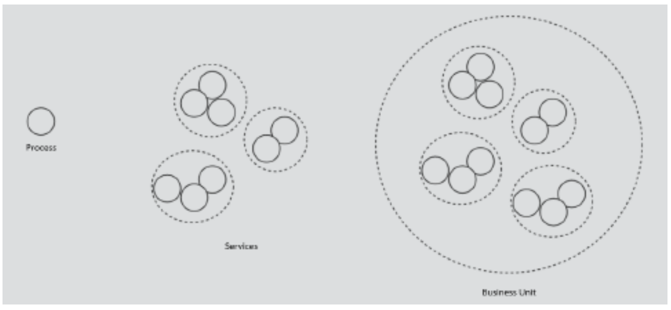


Fig. 1. Datacenter workload groupings.

Processes are grouped based on the services provided, leading to constraints from business units as illustrated in Fig 1. These processes align with virtual or physical machines. The core of the system is the architectural framework of machines. Each machine has specific RAM and CPU capacities, serving as the computational backbone for the workload's resource demands. This hierarchical arrangement and resource allocation are shown in Fig 2.

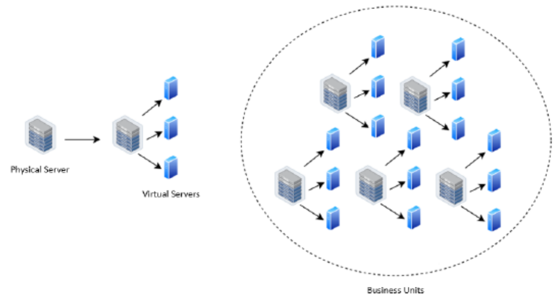


Fig. 2. Datacenter machines grouping.

A physical machine encapsulates the total available capacity (RAM and CPU). For efficiency, security, or redundancy, it can be divided into n virtual machines. The sum of capacities across the virtual machines must not exceed the physical machine's total capacity.

Next, consider migrating workloads from on-premises infrastructure to a public cloud. At defined intervals, specific processes transition from on-premises infrastructure to the cloud, as illustrated in Fig 3. During this process, workloads coexist in both infrastructures, effectively duplicating their presence.

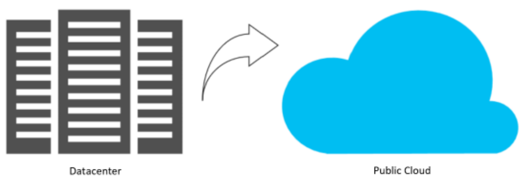


Fig. 3. Workload migration to public cloud from on-premises data center.

The coexistence of processes in both on-premises infrastructure and the public cloud introduces duplicated costs, impacting overall expenses. While parallel existence facilitates migration, the primary objective is to mitigate this dynamic. Implementing a strategy to promptly remove processes from the on-premises data center after successful migration is crucial. This approach allows for a gradual reduction in overall workload, creating opportunities to redistribute processes across machines.

The core problem is efficient resource utilization within the data center, especially given the ongoing migration of IT infrastructure workloads from on-premises to the public cloud. While the literature has explored ways to balance workloads across different IT infrastructures for several years [22]–[27], there has been less focus on strategies that intentionally create imbalances to deactivate idle machines during a parallel migration process.

The goal is to deactivate the maximum number of machines by efficiently redistributing processes to machines with available capacity, enabling the deactivation of underutilized machines. This involves developing resource allocation strategies that ensure effective reorganization, aligning with optimization goals by reducing the data center's capacity and associated costs. Decreasing the number of machines in the on-premises environment during this independent migration process can significantly reduce costs.

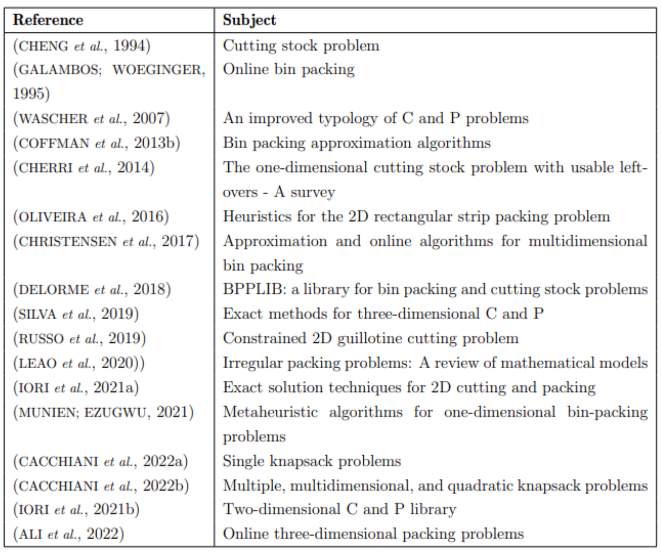
II. Background

In the classical scenario, a list of real numbers within the range (0, 1] is given. The task is to accommodate these numbers in the fewest bins possible, ensuring that the sum of the numbers within each bin does not exceed 1. This problem holds practical significance and finds applications in schedule optimization, route planning, and resource allocation problem-solving. The exploration focuses on the core of this problem, progressively extending the approach to include various generalizations [28]. The bin packing problem is recognized as an NP-hard problem [29], implying that finding an optimal solution becomes computationally infeasible as the problem size increases. In NP-hard problems, there is no known algorithm to solve them efficiently, and their complexity grows rapidly with larger instances. The survey conducted by Christensen [28] provides a historical overview of the key developments in the theory of the bin packing problem, organizing their research according to chronological milestones. Bin packing is acknowledged as a specific variant of the one-dimensional cutting problem [30], the loading problem [31], and several scheduling-related issues [32].

The early stages of this field received significant contributions from other scientists, including Ullman [33]. His work addressed memory allocation problems, such as table formatting and file allocation. Ullman recognized the complexity of finding a general placement algorithm that minimized the number of required bins. Instead, he presented an analysis of two heuristics: FirstFit (FF) and BestFit (BF). Shortly thereafter, Garey and Ullman [34] explored the analysis, considering four heuristics: FirstFit, BestFit, FirstFitDecreasing (FFD), and BestFitDecreasing (BFD). Johnson [35] explored other algorithms and analytical techniques to tackle the challenges of the bin packing problem. Subsequently, himself [36] produced a work that provided a definitive analysis of worst-case guarantees for various approximation algorithms in the context of the bin packing problem.

A. Related Works

In a recent paper by Salem [37], the author conducted a survey on the Covering and Packing Problem, denoted as C and P using their initial letters. In this survey, the author compiled a table that consolidated different works spanning all over the years. Table 2.1 presents a summary of important C and P references. The category of "Cutting and Packing (C and P) problems" encompasses a duo of challenges that share a common foundational structure. These two challenges, despite their distinct names, stem from this common structure. The nomenclature variation arises from the differing application scenarios. Specifically, the "cutting problem" pertains to situations where one or more significant objects necessitate division into a collection of smaller components. In contrast, the "packing problem" relates to scenarios where the objective is to assemble multiple small items into one or more larger objects. Due to the structural similarity, the problems will be collectively referred to as "C and P" without further distinction [37].



Reference	**Subject**
(CHENG *et al.*, 1994)	Cutting stock problem
(GALAMBOS; WOEGINGER, 1995)	Online bin packing
(WASCHER *et al.*, 2007)	An improved typology of C and P problems
(COFFMAN *et al.*, 2013b)	Bin packing approximation algorithms
(CHERRI *et al.*, 2014)	The one-dimensional cutting stock problem with usable left-overs - A survey
(OLIVEIRA *et al.*, 2016)	Heuristics for the 2D rectangular strip packing problem
(CHRISTENSEN *et al.*, 2017)	Approximation and online algorithms for multidimensional bin packing
(DELORME *et al.*, 2018)	BPPLIB: a library for bin packing and cutting stock problems
(SILVA *et al.*, 2019)	Exact methods for three-dimensional C and P
(RUSSO *et al.*, 2019)	Constrained 2D guillotine cutting problem
(LEAO *et al.*, 2020))	Irregular packing problems: A review of mathematical models
(IORI *et al.*, 2021a)	Exact solution techniques for 2D cutting and packing
(MUNIEN; EZUGWU, 2021)	Metaheuristic algorithms for one-dimensional bin-packing problems
(CACCHIANI *et al.*, 2022a)	Single knapsack problems
(CACCHIANI *et al.*, 2022b)	Multiple, multidimensional, and quadratic knapsack problems
(IORI *et al.*, 2021b)	Two-dimensional C and P library
(ALI *et al.*, 2022)	Online three-dimensional packing problems

Fig. 4. References on Cutting and Packing Problems [37]

As elucidated by [37], the resolution of these problems carries significant importance, extending beyond a scientific challenge due to its inherent complexity. It also holds substantial economic significance by contributing to the reduction of a primary cost factor in numerous production sectors. The raw materials are the target of these production sectors and can constitute a substantial portion, sometimes up to 40%, of the overall production expenses.

These considerations are applicable to the data center scenario because maintaining this type of structure is monetarily expensive, involving costs for maintenance, usage, and machine substitutions. It is important to note that there is an economic risk in the problem described above. If the goal is to decommission a data center, but the process is not executed properly, it could lead to a scenario where a company has to bear the costs of two infrastructures simultaneously - one on-premises and the other in the public cloud.

III. Model's Characteristics

We now introduce the set of all items, denoted as I. Each of these items in set I is unique and has certain characteristics described by a pair of values, denoted as (d_{1i}, d_{2i}). In our application, these two positive integers represent its RAM and CPU demands, as shown in 1.

$$\forall i \in I : \exists d_{1i}, d_{2i} \in Z^+ : i = (d_{1i}, d_{2i}) \tag{1}$$

Items will be classified into three distinct groups: i_l, i_b, and i_c, with each group comprising items that adhere to similar characteristics (not necessarily related only to RAM and CPU demands).

The concept of item groupings is defined as any subset $S_n \subseteq I$, where there exists an unordered list of elements $(i_1, i_2, \ldots, i_n)$ such that $S_n = \{i_1, i_2, \ldots, i_n\}$.

$$\forall S_n \subseteq I, \exists i_1, i_2, \ldots, i_n \in I : S_n = \{i_1, i_2, \ldots, i_n\} \tag{2}$$

In task management, each task has two key requirements: CPU processing power and RAM. Tasks can be grouped into three categories: those requiring more CPU ("CPU tasks"), more RAM ("RAM tasks"), or both ("CPU+RAM tasks"). These groupings help organize tasks with similar CPU or RAM needs.

In the "Vector Bin Packing"problem, CPU and RAM requirements correspond to the dimensions of vectors that must be allocated to bins, ensuring the capacity constraints of each bin are respected.

The grouping described earlier is an intermediate classification. A data center provides computational resources to support applications and digital products, managed financially as part of the business. In this scenario, business units fund the acquisition, maintenance, and use of machines in the data center. For governance and security, task groups (CPU tasks, RAM tasks, CPU+RAM tasks) cannot be allocated to machines belonging to other business units. To address this, we define another grouping that links business units to their respective task sets. Let $SC_n \subseteq P(I)$, where $P(I)$ is the power set of I. For each $SC_n \subseteq P(I)$, there is an unordered list $(S_1, S_2, \ldots, S_m)$ of subsets of I, where:

$$\forall SC_n \subseteq P(I), \exists S_1, S_2, \ldots, S_n \subseteq I : SC_n = S_1 \cup S_2 \cup \cdots \cup S_n \tag{3}$$

As an example, consider the sets:

$$S_3 = \{i_{13}, i_2, i_9\}, \quad S_7 = \{i_5, i_{87}\} \tag{4}$$

where:

$$i_{13} = (10, 4), \quad i_2 = (4, 8), \quad i_9 = (20, 2) \tag{5}$$

and

$$i_5 = (2, 2), \quad i_{87} = (2, 1) \tag{6}$$

Now, consider the grouping SC_1:

$$SC_1 = \{S_3, S_7\} = \{(10, 4), (4, 8), (20, 2)\} \cup \{(2, 2), (2, 1)\} \tag{7}$$

$$SC_1 = \{(10, 4), (4, 8), (20, 2), (2, 2), (2, 1)\} \tag{8}$$

The corresponding d_1 and d_2 values can be summed to calculate the resources required by the final grouping SC_1:

$$R_{SC_1} = (10+4+20+2+2, 4+8+2+2+1) = (38, 17) \tag{9}$$

The notation $R(SC_n)$ represents the resource demand of a grouping.

The problem revolves around packing a set of items into a collection of bins. To achieve this, an understanding of what these bins represent is initially required. In this context, a bin is defined as a pair of positive integer values, denoted as $m = (c_1, c_2)$. Thus the bins m in the set B of all bins:

$$\forall m \in B, \exists c_{1i}, c_{2i} \in Z^+ : m = (c_{1i}, c_{2i}) \tag{10}$$

In the allocation process, groups of bins may be relevant. Therefore, we define a bin grouping L_m as a subset of bins:

$$\forall L_m \subseteq B, \exists m_1, m_2, \ldots, m_n \in B : L_m = \{m_1, m_2, \ldots, m_n\} \tag{11}$$

Thus, a bin grouping can be represented by an unordered list of individual bins $(m_1, m_2, \ldots, m_n)$. L_m, as a subset, shares similarities with SC_n groups. The sum of c_1 and c_2 values in any unordered list within L_m allows for the assessment of required resources, denoted by $R(L_m)$. The distinction is that S_n requires capacity from L_m.

A. Model's Interaction

The model distinguishes between different classes of items, denoted as i_l, i_c, and i_b, each subject to specific allocation and reallocation conditions. Items and bins are categorized based on data center management criteria. Critical processes (i_b) pose an allocation challenge due to their complexity and the risk of disrupting business operations. The number of dependencies between items and bins measures the complexity of the allocation process. Critical processes (i_b) have more dependencies, making their allocation difficult. In contrast, processes with fewer dependencies (i_l) allow for simpler allocations. Suppressed demands (i_c) wait in a queue until sufficient capacity is available, after which they can be allocated.

Items of classes i_l and i_c can be allocated to bins in set B_l, with their allocation status adjustable in subsequent iterations. Items of class i_b are bound to bins in set B_b and cannot be reallocated.

Let B_l be the set of bins for items of classes i_l and i_c.

Let B_b be the set of bins for items of class i_b.

In any application, it is necessary to define the constraints for allocating and reallocating items to bins.

Items in S_n can belong to any class of I, meaning they can be part of S_n if $i \in i_l, i_b$, or $i \in i_c$. The allocation rules use the concept of item groupings, represented by S_n, which contains items from classes i_l, i_c, and i_b. Allocation is constrained by the classifications of item and bin groupings. Items can only be allocated to bins if their classifications match, ensuring compatibility.

Constraints are enforced by allocation functions $f : SC_n \to L_m$, where SC_n is an item grouping with classification c_{SC_n} and L_m is a bin grouping with classification c_{L_m}. The function f is defined as:

$$f(SC_n) = L_m \quad \text{if and only if} \quad c_{SC_n} = c_{L_m} \tag{12}$$

B. Constraints

Managing bin capacities is essential in bin packing. We introduce the Bin Capacity Constraints, which govern the allocation of items based on two key attributes: d_{1i} and d_{2i}.

$$\sum_{i \subseteq SC_n} d_{1i} \leq \sum_{m \in L_m} c_{1m} \tag{13}$$

This ensures that the total d_1 of items in S_n doesn't exceed bin capacity c_1 in L_m.

$$\sum_{i \subseteq SC_n} d_{2i} \leq \sum_{m \in L_m} c_{2m} \tag{14}$$

This mirrors the above constraint but applies to the second attribute d_2.

Denoted $CB_{cm} \in [0, 1]$, reserves a portion of bin capacity for class i_c.

$$\sum_{i \subseteq SC_n} d_{1i} \leq (1 - CB_{cm}) \cdot c_{1m} \tag{15}$$

Ensures the first attribute sum for class i_c items stays within reserved capacity limits.

$$\sum_{i \subseteq SC_n} d_{2i} \leq (1 - CB_{cm}) \cdot c_{2m} \tag{16}$$

Similar to last constraint, but for the second attribute.

Reallocation depends on item and bin classes. Items from i_l or i_c can be reallocated within B_l, while items from i_b in B_b cannot be reallocated. Machine capacity reservation (1%-5%) for class i_c and resilience (10%-20%) is enforced before the allocation process.

C. Metrics and Allocation Strategy

Let A be a binary matrix representing the allocation status of items within bins. The entries A_{ij} are defined as:

$$A_{ij} = \begin{cases} 1, & \text{if item } i \text{ is allocated to bin } j \\ 0, & \text{otherwise} \end{cases}$$

This binary encoding encapsulates the allocation decisions, ensuring each item i is exclusively allocated to one bin, adhering to:

$$\forall i \in I, \sum_{j \in B} A_{ij} = 1 \tag{17}$$

where B represents the set of all bins. This constraint ensures the integrity of the allocation process.

Considering the bins as column vectors in A, we partition A as:

$$A = [a_1, a_2, \ldots, a_m] \tag{18}$$

where each a_j denotes a column vector representing a bin. The L_1-norm of a column, $\|a_j\|_1$, measures the bin occupation in terms of the number of items:

$$\|a_j\|_1 = \sum_{i=1}^{s} |a_{ij}| \tag{19}$$

The allocation strategy involves identifying and removing empty bins from A. The algorithm then targets the column with the minimum sum, reallocating items associated with that bin.

IV. Algorithm

This section will present a pseudo-code algorithm specifically designed to address the item allocation problem, which is the focus of this work. The algorithm serves as a computational tool applied to real-world scenarios related to bin packing in the context of data center decommissioning.

The provided code introduces a heuristic allocation approach applied to the bin packing problem. This algorithm incorporates elements of the quicksort algorithm. QuickSort [38], [39] is a widely used sorting algorithm known for its efficiency and speed in sorting elements in a list or array. Furthermore, the proposed heuristics integrates concepts from the first-fit decreasing strategy [29], [33], [36], heuristic method commonly used in bin packing scenarios.

In this context, first-fit decreasing involves sorting items in descending order based on size and sequentially allocating them to the first bin with sufficient space. This strategic combination increases the algorithm's ability to allocate items within the constraints of the packing problem.

At the core of this algorithm lies the sorting of the bins and items in lists, a common strategy employed in bin packing for efficiently allocating items into bins. By leveraging these combined techniques, the algorithm makes more informed and effective decisions during the allocation process. The pseudocode provided further illustrates these concepts.

Algorithm 1 Item Allocation Pseudocode

```
Input: List of items $i$ and list of bins $m$
Output: Allocated bins and items
1  Sort both lists $i$ and $m$ in descending order based on their capacities
2  while the maximum number of items is not allocated to the bins do
3      for each item in $i$ do
4          for each bin in $m$ do
5              if the CPU and RAM capacities of bin in $m$ are greater than or equal to item's demand then
6                  Start allocation
7              else
8                  Proceed to the next bin
9  Generate two new lists for the items Result: items_fitted: A list of items that are allocated to the bins
   Result: items_notfitted: A list of items that could not be allocated to any bin
10 Generate two new lists for the bins Result: bins_fitted: A list of bins that are fully filled after the allocation
   Result: bins_notfitted: A list of bins that still have remaining capacity after the allocation
11 for each bin in bins_fitted do
12     Calculate the remaining capacity information for each bin by subtracting the demand of each item from items_fitted
13 Create the matrix $A$ based on the list of items in items_fitted and the bins in bins_fitted
14 Calculate the L1-norm (norm_one) of the columns of matrix $A$
15 Remove columns with zero norm and choose which bins (i.e., which columns in $A$) will have items removed and/or reallocated
16 for each item in list $i$ do
17     Some items will be removed, which may exist in both items_fitted and items_notfitted
18 Repeat from step 3
19 Stopping condition: If the norm is equal to 0 or there is no possibility of further allocation of items into bins, break the loop
```

When data is grouped and the approach described in Algorithm 1 is executed, it is possible to automate the resource allocation process in a data center environment. This process provides timely and accessible information to meet decision-making needs in specific situations, covering cases of small, medium, and large volumes of data.

A. Algorithm Complexity Analysis

Assessing the algorithm's performance is crucial when considering its utility in practical applications. Algorithm 1 integrates two heuristics with distinct computational complexities during execution: sorting items and bins, and allocating items in bins using first fit decreasing.

1) Sorting Complexity: In its initial phase, the algorithm necessitates sorting the lists of items and bins. This sorting operation is executed through the $sort - values$ method from the $pandas$ library in $Python$. Pandas is a powerful data manipulation library that provides data structures like DataFrames, and the $sort - values$ method efficiently implements sorting algorithms, such as $quicksort$ or $mergesort$. Mergesort is a stable, comparison-based sorting algorithm known for its consistent performance and reliable results.

The expected time complexity of quicksort for n elements is $O(nlogn)$. Considering the number of items and bins denoted as n and m respectively for complexity considerations, the sorting complexities can be expressed as follows: sorting of items $O(nlogn)$ and sorting of bins: $O(mlogm)$.

2) First-Fit Decreasing Complexity: At its core, the algorithm iterates over each item in the list (a total of n iterations). For each item, the algorithm iterates over bins that satisfy the constraint, leading to a maximum of m iterations. Consequently, the overall iteration complexity is expressed as $O(n \times m)$.

Calculating the overall complexity involves summing up these components:

Total Complexity = Sorting of Items + Sorting of Bins + Iteration Complexity.

Total Complexity = $O(nlogn) + O(mlogm) + O(n \times m)$.

The actual runtime of the algorithm depends on the relative sizes of n and m. If both are large and comparable, the dominant factor will be the iteration complexity $O(n \times m)$. However, in scenarios where one is significantly smaller than the other, the dominant factor may vary. Assessing the algorithm's real-world performance requires considering these complexities in the context of specific input data.

V. Conclusion

The experiments are conducted using various items and bins, each characterized by their RAM and CPU demands. The algorithm implementation was on Python, the system used features an x86-64 architecture, 8 CPU cores, with an Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz, and 16GB of memory (RAM). The experiments assess the efficiency of different allocation algorithms in terms of resource usage and allocation accuracy. The results highlight the strengths and limitations of our approach, providing insights for future improvements.

Experiment	Parameters	First Iteration
1	20 Bins, 100 Items	0.14s (Worst Norm: 10)
2	200 Bins, 1,000 Items	2.7s (Worst Norm: 12)
3	2,000 Bins, 10,000 Items	147s (Worst Norm: 7)
4	20,000 Bins, 100,000 Items	14113s (Worst Norm: 16)

TABELA I

Performance metrics (part 1).

Last Iteration	Worst Average Time	Number of Simulations
0.008s (Best Norm: 1)	0.12s	100
0.1s (Best Norm: 1)	2.83s	100
4.7s (Best Norm: 1)	248	30
388s (Best Norm: 1)	13452s	8

TABELA II

Performance metrics (part 2).

The algorithms struggle to perform well on large datasets, as evidenced by the number of parameters used in Experiment 4. In practice, this could be a problem in a data-center with dynamic workloads, where long processing times for resource allocation and migration are unacceptable. Another challenge arises when integrating data centers and cloud infrastructures.

The algorithm's ability to make calculated suggestions for resource allocation and process migration was evidenced. However, the time taken to perform calculations and to migrate processes to their new machines is significant. This is where cost becomes a central factor. If the process relocation and workload migration do not meet the organization's dynamic requirements, the organization could be stuck with two expensive infrastructures: the cloud and on-premises datacenter.

In conclusion, this study advances research on decommissioning data centers during cloud migrations and dynamic workloads. Future work should explore practical solutions, test various heuristics, and include monetary factors for a comprehensive economic analysis. Adding dynamic elements like real-time workload adjustments can enhance model adaptability. As multi-cloud strategies become more common, future research should address decommissioning challenges across multiple public clouds and consider new constraints to optimize resource allocation.

Referências

[1] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. New York: Dover Publications, 1982.

[2] E. G. Coffman, J. Csirik, G. Galambos, S. Martello, and D. Vigo, *Bin Packing Approximation Algorithms: Survey and Classification*. New York: Springer, 2013.

[3] D. Marinescu, *Cloud Computing. Theory and Practice*. Morgan Kaufmann, 2022.

[4] G. Boss, P. Malladi, D. Quan, L. Legregni, and H. Hall, "Cloud computing," *IBM white paper*, vol. 321, pp. 224–231, 2007.

[5] L. Qian, Z. Luo, Y. Du, and L. Guo, "Cloud computing: An overview," in *Cloud Computing: First International Conference, CloudCom 2009, Beijing, China, December 1-4, 2009. Proceedings 1*. Springer, 2009, pp. 626–631.

[6] H. C. Lim, S. Babu, J. S. Chase, and S. S. Parekh, "Automated control in cloud computing: challenges and opportunities," in *Proceedings of the 1st workshop on Automated control for datacenters and clouds*, 2009, pp. 13–18.

[7] Y. Wei and M. B. Blake, "Service-oriented computing and cloud computing: Challenges and opportunities," *IEEE Internet Computing*, vol. 14, no. 6, pp. 72–75, 2010.

[8] K. Al Nuaimi, N. Mohamed, M. Al Nuaimi, and J. Al-Jaroodi, "A survey of load balancing in cloud computing: Challenges and algorithms," in *2012 second symposium on network cloud computing and applications*. IEEE, 2012, pp. 137–142.

[9] F. F. Moghaddam, M. Ahmadi, S. Sarvari, M. Eslami, and A. Golkar, "Cloud computing challenges and opportunities: A survey," in *2015 1st international conference on telematics and future generation networks (TAFGEN)*. IEEE, 2015, pp. 34–38.

[10] J. Nayak, B. Naik, A. Jena, R. K. Barik, and H. Das, "Nature inspired optimizations in cloud computing: applications and challenges," *Cloud computing for optimization: Foundations, applications, and challenges*, pp. 1–26, 2018.

[11] L. Abualigah and A. Diabat, "A novel hybrid antlion optimization algorithm for multi-objective task scheduling problems in cloud computing environments," *Cluster Computing*, vol. 24, no. 1, pp. 205–223, 2021.

[12] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," pp. 68–73, 2008.

[13] A. Beloglazov, R. Buyya, Y. C. Lee, and A. Zomaya, "A taxonomy and survey of energy-efficient data centers and cloud computing systems," *Advances in computers*, vol. 82, pp. 47–111, 2011.

[14] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future generation computer systems*, vol. 28, no. 5, pp. 755–768, 2012.

[15] C. Ward, N. Aravamudan, K. Bhattacharya, K. Cheng, R. Filepp, R. Kearney, B. Peterson, L. Shwartz, and C. C. Young, "Workload migration into clouds challenges, experiences, opportunities," in *2010 IEEE 3rd International Conference on Cloud Computing*. IEEE, 2010, pp. 164–171.

[16] K. Ye, X. Jiang, D. Huang, J. Chen, and B. Wang, "Live migration of multiple virtual machines with resource reservation in cloud computing environments," in *2011 IEEE 4th International Conference on Cloud Computing*. IEEE, 2011, pp. 267–274.

[17] J. Zheng, T. S. E. Ng, and K. Sripanidkulchai, "Workload-aware live storage migration for clouds," in *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, 2011, pp. 133–144.

[18] J. Banerjee, "Moving to the cloud: Workload migration techniques and approaches," in *2012 19th International Conference on High Performance Computing*. IEEE, 2012, pp. 1–6.

[19] S. Imai, T. Chestna, and C. A. Varela, "Elastic scalable cloud computing using application-level migration," in *2012 IEEE Fifth International Conference on Utility and Cloud Computing*. IEEE, 2012, pp. 91–98.

[20] I. Mohiuddin and A. Almogren, "Workload aware vm consolidation method in edge/cloud computing for iot applications," *Journal of Parallel and Distributed Computing*, vol. 123, pp. 204–214, 2019.

[21] A. Gupta and S. Namasudra, "A novel technique for accelerating live migration in cloud computing," *Automated Software Engineering*, vol. 29, no. 1, p. 34, 2022.

[22] S. Kumaraswamy and M. K. Nair, "Bin packing algorithms for virtual machine placement in cloud computing: a review," *International Journal of Electrical and Computer Engineering*, vol. 9, no. 1, p. 512, 2019.

[23] A. Fatima, N. Javaid, T. Sultana, W. Hussain, M. Bilal, S. Shabbir, Y. Asim, M. Akbar, and M. Ilahi, "Virtual machine placement via bin packing in cloud data centers," *Electronics*, vol. 7, no. 12, p. 389, 2018.

[24] A. Wolke, B. Tsend-Ayush, C. Pfeiffer, and M. Bichler, "More than bin packing: Dynamic resource allocation strategies in cloud data centers," *Information Systems*, vol. 52, pp. 83–95, 2015.

[25] M. A. Kaaouache and S. Bouamama, "Solving bin packing problem with a hybrid genetic algorithm for vm placement in cloud," *Procedia Computer Science*, vol. 60, pp. 1061–1069, 2015.

[26] N. Aydın, İ. Muter, and Ş. İ. Birbil, "Multi-objective temporal bin packing problem: An application in cloud computing," *Computers & Operations Research*, vol. 121, p. 104959, 2020.

[27] S. Jangiti and S. Sriram. V.S., "Scalable and direct vector bin-packing heuristic based on residual resource ratios for virtual machine placement in cloud data centers," *Computers Electrical Engineering*, vol. 68, pp. 44–61, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045790617312168

[28] H. Christensen, A. Khan, S. Pokutta, and P. Tetali, "Approximation and online algorithms for multidimensional bin packing: A survey," *Computer Science Review*, vol. 24, 01 2017.

[29] M. R. Garey, D. S. Johnson, and M. R. Garey, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H.Freeman and Co Ltd, 1979.

[30] P. C. Gilmore and R. E. Gomory, "A linear programming approach to the cutting-stock problem," *Operations Research*, vol. 9, no. 6, pp. 849–859, 1961.

[31] S. Eilon and N. Christofides, "The loading problem," *Management Science*, vol. 17, no. 5, pp. 259–268, 1971. [Online]. Available: https://doi.org/10.1287/mnsc.17.5.259

[32] R. W. Conway, ""theory of scheduling"," *Addison Wesley*, 1967.

[33] J. D. Ullman, *The performance of a memory allocation algorithm*. Princeton University, Princeton, N.J, 1971.

[34] M. R. Garey, R. L. Graham, and J. D. Ullman, "Worst-case analysis of memory allocation algorithms," *Proceedings of the fourth annual ACM symposium on Theory of computing*, 1972. [Online]. Available: https://api.semanticscholar.org/CorpusID:26654056

[35] D. S. Johnson, "Near-optimal bin packing algorithms, (ph.d. thesis),," Ph.D. dissertation, Massachusetts Institute of Technology - MIT, Cambridge, 1973.

[36] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham, "Worst-case performance bounds for simple one-dimensional packing algorithms," *SIAM Journal on Computing*, vol. 3, no. 4, pp. 299–325, 1974.

[37] K. H. Salem, E. Silva, and J. F. Oliveira, "Cutting and packing problems under uncertainty: literature review and classification framework," *International Transactions in Operational Research*, vol. 30, no. 6, pp. 3329–3360, 2023.

[38] C. A. Hoare, "Algorithm 65: Find," *Communications of the ACM*, vol. 4, no. 7, pp. 321–322, 1961.

[39] C. A. R. Hoare, "Algorithm 64: Quicksort," *Communications of the ACM*, vol. 4, no. 7, p. 321, 1961.