Hindawi
Discrete Dynamics in Nature and Society
Volume 2020, Article ID 1072485, 9 pages
https://doi.org/10.1155/2020/1072485

Research Article

Dispatching-Rule Variants Algorithms for Used Spaces of Storage Supports

Hani Alquhayz,[1] **Mahdi Jemmali**,[1,2] **and Mohammad Mahmood Otoom**[1]

[1]*Department of Computer Science and Information, College of Science at Zulfi, Majmaah University, Al-Majmaah 11952, Saudi Arabia*
[2]*Department of Computer Science, Higher Institute of Computer Science and Mathematics of Monastir, Monastir University, Monastir 5000, Tunisia*

Correspondence should be addressed to Mahdi Jemmali; m.jemmali@mu.edu.sa

Received 19 December 2019; Revised 8 May 2020; Accepted 23 May 2020; Published 13 June 2020

Academic Editor: J. R. Torregrosa

The paper is regarding the fair distribution of several files having different sizes to several storage supports. With the existence of several storage supports and different files, we search for a method that makes an appropriate backup. The appropriate backup guarantees a fair distribution of the big data (files). Fairness is related to the used spaces of storage support distribution. The problem is how to find a fair method that stores all files on the available storage supports, where each file is characterized by its size. We propose in this paper some fairness methods that seek to minimize the gap between used spaces of all storage supports. In this paper, several algorithms are developed to solve the proposed problem, and the experimental study shows the performance of these developed algorithms.

1. Introduction

The manner that ensures the storage process is a primordial issue. Indeed, facing big data and few storage supports, it is important to seek a method that stores all given files in these storage supports. These methods must guarantee an equity distribution of the files in the available storage supports. We mention that the equity distribution can be called file size balancing. The file size balancing depends on some scheduling algorithms to guarantee a minimum gap between the used spaces of the storage supports. In the nonbalancing case, we face a problem where some storage supports have high used space, and at the same time, some storage supports probably have low used space. To avoid these cases, appropriate scheduling algorithms can be applied. In this paper, we proposed some balancing algorithms to obtain an equity distribution of the files to the storage supports, as far as we know this problem is never studied in the literature review. Some research works related to the balancing process can be cited. Singh et al. [1] proposed a dynamic load balancing algorithm of strongly connected servers, which takes into account these servers capability of parallel processing and their request queuing capacity in order to classify the overloaded servers and the least loaded servers, and once a server gets overloaded, its load is migrated to the least loaded one. In [2], Hung et al. introduced an enhancement of the max-min scheduling algorithm by decreasing the completion time of the clients' requests; this algorithm uses a "supervised machine learning" that clusters utilization percent of virtual machines and clusters size of requests, and then the virtual machine which has the least utilization percent is appointed to the largest cluster requests. However, another work innovated new strategy increases the utilization of virtual machines in an efficient way. This strategy is a heuristic based on a load balancing algorithm which is applied on infrastructure as a service cloud [3]. Besides, Ragmani et al. [4] devised another new strategy of load balancing to increase cloud performance. On the contrary, another work integrated a load balancing algorithm to resource scheduling to provide a higher quality of cloud service [5]. In [6], Hung et al. proposed a load balancing algorithm called max-min and max algorithm that

computes the average completion time for every task in all nodes, and then the task with the maximum average completion time is dispatched to the unassigned node with the minimum completion time which is less than the task maximum average completion time. The work in [7] focuses on maintaining information about every virtual machine efficiency in an allocation table that resides in a data center, by increasing the allocation count for an efficient virtual machine which has been allocated by a request and decreasing its allocation count after completing that request. Some other works apply balancing algorithms to solve real-life problems. Indeed, Jemmali et al. [8, 9] treated the problem of gas turbine aircraft engines. However, Jemmali [10] focused on the equity distribution of projects revenues assignment. In the latter work, authors proposed several approximate solutions to solve the problem. Several other works treated the problem of balancing in different applications. Hasan et al. [11] applied balancing algorithm on small-cell networks to adjust handover parameters of the overloaded cells with adjacent cells. However, the balancing algorithm was applied on voltage loads of capacitors in a modular multilevel converter [12].

Xu et al. [13] introduced a technique that rewrites data blocks and defragments the backups of VM images as well as the authors proposed a technique for restoration of VM image backups by caching these data blocks. However, in [14], Xu et al. proposed a method based on enhanced k-means clustering that finds VM images of duplicated segments to be selected, and then these VM images can be loaded into memory.

Jain and You and Koseki and Ogawa [15, 16] proposed a method of load balancing of a set of nodes within the cluster storage system. This method identifies a source node and target node based on a threshold value of the load as well as a proximity between the source and target nodes. This method chooses the data objects to be moved from the source node to the target node without exceeding the threshold value of load in the target node after moving these data objects.

Besides, Gulati et al. [17] introduced a software system which handles the placement of virtual machines and implement load balancing between several devices automatically by applying migrations of data between devices and without the need to the storage arrays.

In [18], Hu et al. introduced a load balancing strategy of VM resources using genetic algorithm and the system previous data and its current state. This strategy selects the best load balancing and alleviates or gets rid of dynamic migration.

However, Aerts et al. [19] proposed models of load balancing that can be done based on NP-hard retrieval time as well as blocks basis.

In this paper, our study focused on distributed load balancing algorithms, due to using the centralized algorithms limit the scalability in the future as well as they make the system less fault tolerable. Besides, our algorithms deal with a batch of files that need to be stored in temporary storage, and depending on the system planned backup time T, the load balancer is triggered. Therefore, number of files is not a matter.

Table 1: Size files values.

f_j	f_1	f_2	f_3	f_4	f_5	f_6	f_7
s_j	8	3	10	5	11	7	13

This paper is organized as follows. In Section 2, we present the studied problem and we give some details about the problem in general. Section 3 presents six proposed algorithms for the studied problem, and the experimental results are presented in Section 4.

2. Problem Definition

The problem studied in this paper is the proposition of the fairness method that guarantees the fair distribution of several files to the storage supports. The problem can be presented as follows. Let F be the set of given files that must be stored on a fixed number of storage supports. The number of files is denoted by n_f and the number of storage supports is denoted by n_s. The set of storage supports is denoted by $\mathrm{ST} = \{\mathrm{st}_1, \cdots, \mathrm{st}_{n_s}\}$. Each file f_j with $j = \{1, \cdots, n_f\}$ is characterized by its size s_j. When the file f_j is stored, the cumulative file size is denoted by cs_j. The total used space for the storage support st_i when all files are stored is denoted by Us_i with $i = \{1, \cdots, n_s\}$. The minimum (maximum) used space after the termination of the backup procedure is denoted by $Us_{\min}$ ($Us_{\max}$). Example 1 can illustrate the studied problem.

Example 1. Let $n_f = 7$ and $n_s = 2$. Table 1 represents the sizes for each file f_j.

We seek to store the seven files on the two given storage supports. Applying an algorithm rule, the result is given in Figure 1.

The results given by the scheduling shown in Figure 1 are as follows. We store the files {2, 6, 3, 7} in storage support 1 and files {4, 5, 1} will be stored in storage support 2. Based on the latter schedule, the used space for storage support 1 is 33. However, storage support 2 has a used space of 24. The gap between storage support 1 and storage support 2 is $Us_1 - Us_2 = 9$. Seeking to reduce the latter gap is our primordial objective in this research work. Thus, we must search for a schedule that reduces the gap with a more efficient value less than 9.

3. Approximate Solutions

Our objective in this paper is to minimize the gap between storage supports. To do that, we must, in the first step, define the gap in a general case. The gap can be calculated using different methods. We propose the indicator that can calculate the gap as follows: for each storage support, we subtract the minimum value of all used spaces from the used space of the corresponding storage support. Therefore, considering the n_s storage supports, the total capacity gap (TC_g) is given by the following equation:

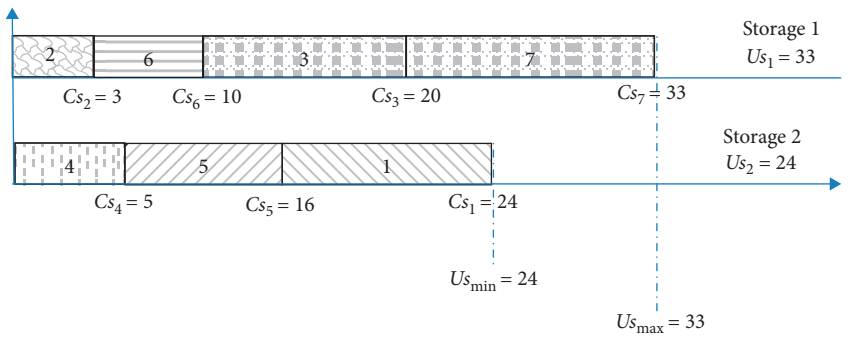

Figure 1: Size file-storages scheduling.

(1) Set $j = 1$, $P = F$
(2) **while** $(j \leq n_f)$ **do**
(3) **if** $(\text{Mod}(j/2) = 1)$ **then**
(4) $k = \text{NIS}(P)$
(5) **else**
(6) $k = \text{NDS}(P)$
(7) **end if**
(8) $\text{Store}(k)$
(9) $P = P \backslash k$
(10) $j++$
(11) **end while**

Algorithm 1: Swapping nonincreasing-decreasing algorithm: SIDA.

$$\text{TC}_g = \sum_{i=1}^{n_s} [Us_i - Us_{\min}]. \quad (1)$$

Our objective is to minimize TC_g given in equation (1).

Based on the standard three-field notation in [20], the studied problem is denoted by $P \| \text{TC}_g$.

Proposition 1. *The $P \| TC_g$ is an NP-hard problem.*

Proof. Since $P \| Us_{\min}$ is NP-hard problem [21] the studied problem is NP-hard problem because $\text{TC}_g = \sum_{i=1}^{n_s} [Us_i - Us_{\min}]$.

To achieve the goal of the work, we propose several algorithms to give approximate solutions.

The proposed algorithms in this paper are based on three methods to solve the studied problems: the first method used the dispatching rules (nonincreasing sizes order algorithm (NISA) and nondecreasing sizes order algorithm (NDSA)), the second method is based on swapping approach (swapping nonincreasing-decreasing sizes order algorithm (SIDA), swapping nondecreasing-increasing sizes order algorithm (SDIA)), the third type of method is more complicated and is based on a mixed approach between the largest files and smallest ones (swapping nonincreasing-decreasing sizes with order algorithm (SIDA^r), r-Swapping nondecreasing-increasing sizes with order algorithm (SDIA^r)). □

3.1. Nonincreasing Sizes Order Algorithm (NISA). This algorithm is applied since all files are ordered by the nonincreasing order of its sizes. After that, we store the files which have the greatest size in the storage support that has the minimum used space.

3.2. Nondecreasing Sizes Order Algorithm (NDSA). This algorithm is applied since all files are ordered by the nondecreasing order of its sizes. After that, we store the files which have the smallest size in the storage support that has the minimum used space.

3.3. Swapping Nonincreasing-Decreasing Sizes with Order Algorithm (SIDA). Instead to apply just one order (nonincreasing or nondecreasing), we adopt a mixture one by one. This means that for a first selection, we pick the file which has the largest size and for the second selection, we take the file which has the smallest size and so on until all the files are stored. This algorithm works in fact on swapping between two algorithms. The function that call the algorithm NISA(), is denoted by NIS(), and the function that call the algorithm NDSA is denoted by NDS(). These two functions return the file index that satisfies the applied algorithm. The function $\text{Store}(j)$ is responsible to store the file f_j in the most available storage supports. The most available storage supports are the storage supports which have the minimum used space. The algorithm of SIDA is given in Algorithm 1.

3.4. Swapping Nondecreasing-Increasing Sizes with Order Algorithm (SDIA). This algorithm is based on the same idea of SIDA. The difference here is instead of beginning with the file that has the largest size, we inverse it and begin with the file that has the smallest size. The second file will be the one that has the largest size and so on.

The algorithm of SDIA is given in Algorithm 2.

```
(1) Set j = 1, P = F
(2) while (j ≤ n_f) do
(3)    if (Mod(j/2) = 1) then
(4)       k = NDS(P)
(5)    else
(6)       k = NIS(P)
(7)    end if
(8)    Store(k)
(9)    P = P\k
(10)   j++
(11) end while
```

Algorithm 2: Swapping nondecreasing-increasing algorithm: SDIA.

3.5. Swapping Nonincreasing-Decreasing Sizes with Order Algorithm (SIDA^r). This algorithm is based on the following idea. Instead of swapping files by files applied in SIDA, we select once the largest file and once the smallest file. The question is how the algorithm becomes if we go ahead for the swapping by two files or three files or r files.

If we choose the 2-swapping, this means we select the two files having the largest sizes and we store it. After that, we select the two files having the smallest sizes and so on two by two.

The algorithm of 2-swapping, which means that $r = 2$, is given in Algorithm 3.

Algorithm 3 can give the solution just for 2-swapping files. We can generalize Algorithm 3 by searching the solution when $r > 2$. The algorithm for a predetermined $r = t$ is given in Algorithm 4.

The performance and generalization of the abovementioned algorithm are based on the iteration of Algorithm 4 several times, and then we select the best solution. This generalization will be given as Algorithm 5.

3.6. r-Swapping Nondecreasing-Increasing Sizes with Order Algorithm (SDIA^r). This algorithm has the same idea as the abovementioned algorithm. The difference here is instead of starting with the largest files, we start with the smallest files.

In Algorithm 5, we replace NIS(P) by NDS(P) in instruction 4 and we replace NDS(P) by NIS(P) in instruction 14. This modification will give the new algorithm SDIA$_t$. In Algorithm 5, we modify SIDA$_t$ (F) by SDIA$_t$ $()$ in instruction 2, and then the new algorithm SDIA^r will be obtained.

4. Case Study

In this case study, we give the comparison between NISA and all other heuristics expected NDSA because for 100% of cases, NISA is better. So, we show a case study of instances that our proposed algorithms are better than NISA.

4.1. Comparison of NISA and SIDA^r. Let the instance with 10 files be assigned to 2 storage supports. The sizes of the 10 files are given in Table 2.

This case study is given for the first execution of the algorithm NISA. This means the used space is zero for both storage 1 and storage 2.

The first step of the NISA algorithm is ordering the files in a nonincreasing way. This gives the following order of a batch of files $\{f_3, f_8, f_2, f_1, f_7, f_{10}, f_4, f_9, f_6, f_5\}$.

The steps of NISA algorithm are as follows: the largest file f_3 will be placed in the storage support with minimum used space. Storage 1 is selected. Then, the second largest file f_8 is placed in storage 2 which is the one with minimum used space at this point. After that, the third file f_2 will be assigned to storage 2 because it has the minimum used space at this point (storage 1 : 280, storage 2 : 268). Now, the used space in storage 2 is 526 and so on.

Therefore, the schedule given by applying the algorithm *NISA* is as follows: in storage 1, NISA assigned the files $\{f_3, f_1, f_{10}, f_4, f_5\}$. So, the total size assigned to this storage support is 939. However, in storage 2, the algorithm assigned the files $\{f_8, f_2, f_7, f_9, f_6\}$. So, the total size assigned to storage support 2 is 988. Therefore, the gap between storages is

$$\begin{aligned} \mathrm{TC}_g(\mathrm{NISA}) &= \sum_{i=1}^{2} [Us_i - Us_{\min}] = (939 - 939) \\ &\quad + (988 - 939) = 49. \end{aligned} \tag{2}$$

On the contrary, applying the algorithm SIDA^r, the schedule is given as follows: in storage 1, SIDA^r assigned the files $\{f_3, f_5, f_6, f_7, f_9, f_{10}\}$. So, the total size assigned to this storage support is 973. However, in storage 2, the algorithm assigned the files $\{f_1, f_2, f_4, f_8\}$. So, the total size assigned to storage support 2 is 954. Therefore, the gap between storages is

$$\begin{aligned} \mathrm{TC}_g(\mathrm{SIDA}^r) &= \sum_{i=1}^{2} [Us_i - Us_{\min}] = (973 - 954) \\ &\quad + (954 - 954) = 19. \end{aligned} \tag{3}$$

It is clear to observe that g_{SIDA^r} is less than g_{NISA}. Therefore, by comparing the results given by the algorithms NISA and SIDA^r, the difference is 30. Thus, SIDA^r gives the minimum gap.

4.2. Comparison of NISA and SIDA. Let the instances with 10 files be assigned to 2 storage supports. The sizes of the 10 files are given in Table 3.

The schedule given by applying the algorithm NISA is as follows: in storage 1, NISA assigned the files $\{f_7, f_5, f_2, f_{10}, f_4\}$. So, the total size assigned to this storage support is 269. However, in storage 2, the algorithm assigned the files $\{f_6, f_3, f_8, f_9, f_1\}$. So, the total size assigned to storage support 2 is 251. Therefore, the gap between storages is

$$\begin{aligned} \mathrm{TC}_g(\mathrm{NISA}) &= \sum_{i=1}^{2} [Us_i - Us_{\min}] = (269 - 251) \\ &\quad + (251 - 251) = 18. \end{aligned} \tag{4}$$

```
(1) Set j = 1, P = F
(2) while (j ≤ n_f) do
(3)    for (t = 1 to t = 2) do
(4)       k = NIS(P)
(5)       Store(k)
(6)       P = P\k
(7)       j + +
(8)       if (j > n_f) then
(9)          Break;
(10)      end if
(11)   end for
(12)   if (j < n_f) then
(13)      for (t = 1 to t = 2) do
(14)         k = NDS(P)
(15)         Store(k)
(16)         P = P\k
(17)         j + +
(18)         if (j > n_f) then
(19)            Break;
(20)         end if
(21)      end for
(22)   end if
(23) end while
```

Algorithm 3: 2-swapping algorithm: SIDA_2.

```
(1) Set j = 1, P = F
(2) while (j ≤ n_f) do
(3)    for (it = 1 to it = t) do
(4)       k = NIS(P)
(5)       Store(k)
(6)       P = P\k
(7)       j + +
(8)       if (j > n_f) then
(9)          Break;
(10)      end if
(11)   end for
(12)   if (j ≤ n_f) then
(13)      for (it = 1 to it = t) do
(14)         k = NDS(P)
(15)         Store(k)
(16)         P = P\k
(17)         j + +
(18)         if (j > n_f) then
(19)            Break;
(20)         end if
(21)      end for
(22)   end if
(23) end while
```

Algorithm 4: t-swapping algorithm: SIDA_t.

On the contrary, applying the algorithm SIDA, the schedule is given as follows: in storage 1, SIDA assigned the files $\{f_7, f_4, f_9, f_5, f_2\}$. So, the total size assigned to this storage support is 260. However, in storage 2, the algorithm assigned the files $\{f_1, f_6, f_3, f_{10}, f_8\}$. So, the total size assigned to storage support 2 is 260. Therefore, the gap between storages is

$$\mathrm{TC}_g(\mathrm{SIDA}) = \sum_{i=1}^{2}\left[Us_i - Us_{\min}\right] = (260 - 260) + (260 - 260) = 0. \tag{5}$$

It is clear to observe that $\mathrm{TC}_g(\mathrm{SIDA})$ is less than $\mathrm{TC}_g(\mathrm{NISA})$. Therefore, by comparing the results given by the algorithms NISA and SIDA, the difference is 18. Thus, SIDA gives the minimum and optimal gap because $\mathrm{TC}_g(\mathrm{SIDA}) = 0$.

4.3. Comparison of NISA and SDIA. Let the instance with 10 files be assigned to 3 storage supports. The sizes of the 10 files are given in Table 4.

The schedule given by applying the algorithm *NISA* is as follows: in storage 1, NISA assigned the files $\{f_8, f_9, f_7\}$. So, the total size assigned to this storage support is 132. However, in storage 2, the algorithm assigned the files $\{f_1, f_6, f_3, f_4\}$ and the total size assigned to this storage support is 135. For the third storage, the assigned files will be $\{f_{10}, f_2, f_5\}$ and the total size assigned to the latter storage support is 120. Therefore, the gap between storages is

$$\mathrm{TC}_g(\mathrm{NISA}) = \sum_{i=1}^{3}\left[Us_i - Us_{\min}\right] = (132 - 120) + (135 - 120) + (120 - 120) = 27. \tag{6}$$

On the contrary, applying the algorithm SDIA, the schedule is given as follows: in storage 1, SDIA assigned the files $\{f_7, f_1, f_9, f_6\}$. So, the total size assigned to this storage support is 131. However, in storage 2, the algorithm assigned the files $\{f_8, f_2\}$. So, the total size assigned to storage support 2 is 123. In storage 3, SDIA assigned the files $\{f_4, f_3, f_{10}, f_5\}$. So, the total size assigned to this storage support is 131. Therefore, the gap between storages is

$$\mathrm{TC}_g(\mathrm{SDIA}) = \sum_{i=1}^{3}\left[Us_i - Us_{\min}\right] = (133 - 123) + (123 - 123) + (131 - 123) = 18. \tag{7}$$

It is clear to observe that $\mathrm{TC}_g(\mathrm{SDIA})$ is better than $\mathrm{TC}_g(\mathrm{NISA})$. Therefore, by comparing the results given by the algorithms NISA and SDIA, the difference is 9. Thus, SDIA gives the minimum gap.

4.4. Comparison of NISA and SDIA^r. Let the instances with 10 files be assigned to 2 storage supports. The sizes of the 10 files are given in Table 5.

The schedule given by applying the algorithm NISA is as follows: in storage 1, NISA assigned the files $\{f_4, f_8, f_2, f_9, f_5\}$. So, the total size assigned to this storage support is 4961. However, in storage 2, the algorithm assigned

(1) **for** $(t = 2 \text{ to } t = r)$ **do**
(2) $\text{gap}_t = \text{SIDA}_t(F)$
(3) **end for**
(4) $\text{gap} = \min_{2 \le t \le r}(\text{gap}_t)$
(5) Return gap

Algorithm 5: Swapping nondecreasing-increasing algorithm: SIDA^r.

Table 2: Size files values for the case study.

f_j	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
s_j	251	258	280	177	24	91	251	268	120	207

Table 3: Size files values for the case study.

f_j	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
s_j	6	46	82	28	62	92	96	43	28	37

the files $\{f_{10}, f_6, f_7, f_1, f_3\}$. So, the total size assigned to storage support 2 is 4969. Therefore, the gap between storages is

$$\text{TC}_g(\text{NISA}) = \sum_{i=1}^{2}\left[Us_i - Us_{\min}\right] = (4961 - 4961) + (4969 - 4961) = 8. \quad (8)$$

On the contrary, applying the algorithm SDIA^r, the schedule is given as follows: in storage 1, SDIA^r assigned the files $\{f_5, f_9, f_2, f_4, f_6\}$. So, the total size assigned to this storage support is 4965. However, in storage 2, the algorithm assigned the files $\{f_3, f_1, f_7, f_{10}, f_8\}$. So, the total size assigned to storage support 2 is 4965. Therefore, the gap between storages is

$$\text{TC}_g\left(\text{SDIA}^r\right) = \sum_{i=1}^{2}\left[Us_i - Us_{\min}\right] = (4965 - 4965) + (4965 - 4965) = 0. \quad (9)$$

It is clear to observe that $\text{TC}_g(\text{SDIA}^r)$ is better than $\text{TC}_g(\text{NISA})$. Therefore, by comparing the results given by the algorithms NISA and SDIA^r, the difference is 8. Thus, SDIA^r gives the minimum and optimal gap because $\text{TC}_g(\text{SDIA}^r) = 0$.

Inspired from this case study, we proposed to apply our algorithms on a cloud computing domain by adding a new component called "scheduler" in the architecture of the cloud computing. This component will be responsible for applying the proposed algorithms and it gives the optimal schedule.

5. Experimental Results

In this section, we propose different classes of instances to compare the performance of proposed algorithms. The main comparison in this paper is that we compare the developed algorithms with the NISA algorithm. The NISA algorithm is used in the literature review as LPT algorithm which has several applications in the industry.

Table 4: Size files values for the case study.

f_j	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
s_j	45	40	27	24	38	39	19	83	30	42

Table 5: Size files values for the case study.

f_j	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
s_j	990	992	949	1034	937	1013	995	1009	989	1022

Table 6: Size files and storage supports pair choice.

n_f	n_s
10, 20, 25, 50	2, 3, 5
100, 150, 250, 300, 500	2, 3, 5, 10, 15, 20, 25, 30
1000, 1500, 2000, 2500, 3000, 3500	2, 3, 5, 10, 15, 20, 25, 30, 50

The proposed algorithms in this paper were coded and executed by Microsoft Visual C++ (Version 2013). The computer that is utilized to run all programs coded in C++ has the characteristics as follows:

(i) Processor: Intel® Core™ i5-3337U CPU @ 1.8 GHz.
(ii) Operating system: Windows 10.

We adopt the choice that the classes used to discuss the results obtained by the developed algorithms are inspired by the classes proposed in [21].

The generation of the file sizes s_j will be through two kinds of distributions. The unit of the size is proposed as Mo. The studied classes are given as follows:

(i) Class A: s_j is in $U[1, 100]$
(ii) Class B: s_j is in $U[10, 300]$
(iii) Class C: s_j is in $U[500, 1500]$
(iv) Class D: s_j is in $U[50, 250]$
(v) Class E: s_j is in $U[25, 1000]$

U is the uniform distribution and N is the normal distribution. The total number of generated instances depends on the choice of n_f, n_s, and class. The pair (n_f, n_s) can have different permutation values. The determination of the (n_f, n_s) values is given in Table 6.

From Table 6, we can deduce that there are 5300 instances in total. To show the performance of the developed algorithms compared to the oldest LPT one, we can use several indicators. Thus, we propose the following indicators for this paper:

(i) H is the best algorithm returned value after running all heuristics.
(ii) H_d is the discussed heuristic.
(iii) $G_H = ((H_d - H)/H_d)$, if $H_d = 0$, we have GAP = 0.

Table 7: Overall view of algorithm results.

	NISA	NDSA	SIDA	SDIA	SIDA^r	SDIA^r
Per (%)	45.2	0.0	10.6	0.8	75.2	41.0
Time	—	—	—	—	0.115	0.108

Table 8: The behavior of G_H and time when the number of files is changed.

n_f	NISA		NDSA		SIDA		SDIA		SIDA^r		SDIA^r	
	G_H	Time	G_H	Time	G_H	Time	G_H	Time	G_H	Time	G_H	Time
10	0.15	—	0.73	—	0.49	—	0.67	—	0.10	—	0.33	—
20	0.40	—	0.85	—	0.60	—	0.80	—	0.22	—	0.34	—
25	0.36	—	0.82	—	0.79	—	0.58	—	0.24	—	0.44	—
50	0.40	—	0.90	—	0.69	—	0.86	—	0.20	—	0.41	—
100	0.35	—	0.85	—	0.75	—	0.82	—	0.15	0.001	0.34	0.001
150	0.43	—	0.94	—	0.84	—	0.92	—	0.25	0.001	0.31	0.001
250	0.36	—	0.88	—	0.75	—	0.85	—	0.16	0.003	0.33	0.002
300	0.36	—	0.97	—	0.86	—	0.96	—	0.20	0.004	0.36	0.003
500	0.35	—	0.91	—	0.78	—	0.89	—	0.12	0.010	0.39	0.008
1000	0.34	—	0.91	—	0.79	—	0.89	—	0.09	0.042	0.36	0.039
1500	0.36	—	0.98	—	0.88	—	0.98	—	0.14	0.091	0.44	0.092
2000	0.38	—	0.93	—	0.80	—	0.91	—	0.12	0.159	0.38	0.144
2500	0.31	—	0.92	—	0.79	—	0.90	—	0.09	0.250	0.39	0.224
3000	0.35	0.001	0.99	0.001	0.87	—	0.98	0.001	0.14	0.340	0.40	0.319
3500	0.36	0.001	0.93	0.001	0.80	0.001	0.91	0.001	0.10	0.461	0.39	0.444

Table 9: The behavior of G_H and time when the number of storage supports is changed.

n_s	NISA		NDSA		SIDA		SDIA		SIDA^r		SDIA^r	
	G_H	Time	G_H	Time	G_H	Time	G_H	Time	G_H	Time	G_H	Time
2	0.41	—	0.98	—	0.24	—	0.94	—	0.13	0.013	0.46	0.012
3	0.31	—	0.81	—	0.75	—	0.75	—	0.10	0.016	0.37	0.015
5	0.26	—	0.93	—	0.93	—	0.92	—	0.23	0.023	0.28	0.021
10	0.42	—	0.99	—	0.98	—	0.98	—	0.06	0.067	0.50	0.059
15	0.30	—	0.84	—	0.80	—	0.80	—	0.14	0.101	0.31	0.091
20	0.42	—	0.95	—	0.94	—	0.94	—	0.11	0.158	0.47	0.161
25	0.38	—	0.95	—	0.95	—	0.95	—	0.33	0.191	0.21	0.179
30	0.33	—	0.85	—	0.82	—	0.82	—	0.08	0.219	0.36	0.206
50	0.46	0.001	0.99	0.001	0.98	0.001	0.98	0.001	0.11	0.557	0.43	0.517

(iv) Time is the average running time. This time will be in seconds. The symbol "–" means that the time is less than 0.001 s.

(v) Per is the percentage among the total instances (5300) that the constraint $H_d = H$ is obtained.

Several statistics can be presented in this work. We start by the overall Table 7 that shows the percentage of each heuristic when the studied heuristic equals to the best one. The corresponding average time is calculated for each heuristic in Table 7. The algorithm SIDA^r is the best one among all algorithms with percentage 75.2% and average time 0.115 s. However, the algorithm NISA has 45.2% and NDSA has 0%. The algorithm that consumes more running time compared with others is SIDA^r.

Table 8 presents the behavior of G_H and time when the number of files is changed. For all algorithms, when the number of files increases, the Time increases. The worst G_H value which is equal to 0.99 is obtained for the algorithm NDSA when $n_f = 3000$. However, the best gap 0.09 is obtained for algorithm SIDA^r when $n_f = 2500$ (Table 9).

Table 10 represents the behavior of G_H and Time when the number of classes is changed. It is noticed that the worst gap is 0.98 when the NDSA algorithm is applied to class A. On the contrary, the SIDA^r algorithm achieved the best gap in the same class which equals 0.02. The table also shows that the best execution time is less than 0.001s for the algorithms NISA, NDSA, SIDA, and SDIA. However, the algorithms SIDA^r and SDIA^r have the worse execution times which are larger than 0.1s, and the highest execution time is 0.125 s for SIDA^r algorithm when it is applied to class C.

For the algorithm SIDA^r, we can observe that the gap is 0.31 and 0.25 for classes D and E, respectively. However, these values are higher than the gap values of classes A, B, and C which are less than 0.1.

Table 10: The behavior of G_H and time when the number of classes is changed.

Class	NISA		NDSA		SIDA		SDIA		$SIDA^r$		$SDIA^r$	
	G_H	Time	G_H	Time	G_H	Time	G_H	Time	G_H	Time	G_H	Time
A	0.05	—	0.98	—	0.85	—	0.95	—	0.02	0.110	0.53	0.107
B	0.17	—	0.97	—	0.85	—	0.94	—	0.06	0.116	0.43	0.108
C	0.29	—	0.95	—	0.85	—	0.92	—	0.08	0.125	0.44	0.110
D	0.68	—	0.92	—	0.77	—	0.87	—	0.31	0.114	0.26	0.109
E	0.59	—	0.77	—	0.65	—	0.77	—	0.25	0.111	0.22	0.107

6. Conclusion

In this paper, we focused our study on the resolution of the NP-hard problem of assignment of several files to different storage supports. We developed six algorithms to solve the latter problem; these algorithms are essentially based on the dispatching rules with variant methods. These methods are categorized into the nonincreasing (decreasing) order rule and the mixture method that uses both the nonincreasing and decreasing order rules. In addition, we proposed the r-swapping methods which are based on storing the first r files using the nonincreasing rule and then storing the next r files by applying the nondecreasing rule and so on until storing all files. In this paper, we chose the number r which equals $n_f - 1$. The experimental results show that the best algorithm is $SIDA^r$, which outperforms the old algorithms in the literature review. The proposed algorithms can be enhanced to develop more performed new algorithms.

Data Availability

The data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

The authors would like to thank the Deanship of Scientific Research at Majmaah University for supporting this work under Project no. RGP-2019-13.

References

[1] R. Singh, P. K. Gupta, P. Gupta et al., "Load balancing of distributed servers in distributed file systems,"in Proceedings of the International Conference on ICT Innovations, Springer, Ohrid, Macedonia, pp. 29–37, October 2015.

[2] T. C. Hung, L. N. Hieu, P. T. Hy, and N. X. Phi, "MMSIA: improved max-min scheduling algorithm for load balancing on cloud computing," in *Proceedings of the 3rd International Conference on Machine Learning and Soft Computing—ICMLSC 2019*, pp. 60–64, Da Lat, Vietnam, January 2019.

[3] M. Adhikari and T. Amgoth, "Heuristic-based load-balancing algorithm for IaaS cloud," *Future Generation Computer Systems*, vol. 81, pp. 156–165, 2018.

[4] A. Ragmani, A. El Omri, N. Abghour, K. Moussaid, and M. Rida, "A performed load balancing algorithm for public cloud computing using ant colony optimization," *Recent Patents on Computer Science*, vol. 11, no. 3, pp. 179–195, 2018.

[5] V. Priya, C. Sathiya Kumar, and R. Kannan, "Resource scheduling algorithm with load balancing for cloud service provisioning," *Applied Soft Computing*, vol. 76, pp. 416–424, 2019.

[6] C.-L. Hung, H.-H. Wang, and Y.-C. Hu, "Efficient load balancing algorithm for cloud computing network," in *Proceedings of the International Conference on Information Science and Technology (IST 2012)*, pp. 28–30, Hubei, China, March 2012.

[7] M. Sharma, P. Sharma, and S. Sharma, "Efficient load balancing algorithm in VM cloud environment," 2012.

[8] M. Jemmali, L. K. B. Melhim, and M. Alharbi, "Randomized-variants lower bounds for gas turbines aircraft engines," in *Proceedings of the World Congress on Global Optimization*, Springer, Metz, France, pp. 949–956, July 2019.

[9] M. Jemmali, L. K. B. Melhim, S. O. B. Alharbi, and A. S. Bajahzar, "Lower bounds for gas turbines aircraft engines," *Communications in Mathematics and Applications*, vol. 10, no. 3, pp. 637–642, 2019.

[10] M. Jemmali, "Approximate solutions for the projects revenues assignment problem," *Communications in Mathematics and Applications*, vol. 10, no. 3, pp. 653–658, 2019.

[11] M. M. Hasan, S. Kwon, and J.-H. Na, "Adaptive mobility load balancing algorithm for LTE small-cell networks," *IEEE Transactions on Wireless Communications*, vol. 17, no. 4, pp. 2205–2217, 2018.

[12] S. B. Bashir and A. R. Beig, "An improved voltage balancing algorithm for grid connected MMC for medium voltage energy conversion," *International Journal of Electrical Power & Energy Systems*, vol. 95, pp. 550–560, 2018.

[13] J. Xu, T. Wang, X. Xing, W. Zhang, and H. Zhong, "Efficient image restoration of virtual machines with reference count based rewriting and caching," *Future Generation Computer Systems*, vol. 77, pp. 87–96, 2017.

[14] J. Xu, W. Zhang, Z. Zhang, T. Wang, and T. Huang, "Clustering-based acceleration for virtual machine image deduplication in the cloud environment," *Journal of Systems and Software*, vol. 121, pp. 144–156, 2016.

[15] N. Jain and G.-W. You, "Load balancing in cluster storage systems," U.S. Patent no. 8,886,781. 11, 2014.

[16] H. Koseki and J. Ogawa, "Storage apparatus and load balancing method," U.S. Patent no. 8,387,063, 2013.

[17] A. Gulati, C. Kumar, I. Ahmad, and K. Kumar, "BASIL: automated IO load balancing across storage devices," in *Proceedings of the FAST'10: 8th USENIX Conference on File and Storage Technologies*, p. 13, San Jose, CA, USA, February 2010.

[18] J. Hu, J. Gu, G. Sun, and T. Zhao, "A scheduling strategy on load balancing of virtual machine resources in cloud

computing environment," in *Proceedings of the 2010 3rd International Symposium on Parallel Architectures, Algorithms and Programming*, pp. 89–96, IEEE, Dalian, China, December 2010.

[19] J. Aerts, J. Korst, and W. Verhaegh, "Load balancing for redundant storage strategies: multiprocessor scheduling with machine eligibility," *Journal of Scheduling*, vol. 4, no. 5, pp. 245–257, 2001.

[20] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, "Sequencing and scheduling: algorithms and complexity," in *Logistics of Production and Inventory*, vol. 4, pp. 445–522, Elsevier, Amsterdam, Netherlands, 1993.

[21] M. Haouari and M. Jemmali, "Maximizing the minimum completion time on parallel machines," *4OR*, vol. 6, no. 4, pp. 375–392, 2008.