PLOS ONE

RESEARCH ARTICLE

Algorithms for the executable programs planning on supercomputers

Abdullah M. Algashami *

Department of Computer Science and Information, College of Science at Zulfi, Majmaah University, Al-Majmaah, Saudi Arabia

* a.algashami@mu.edu.sa

Abstract

This research dealt with the problem of scheduling applied to the supercomputer's execution. The goal is to develop an appreciated algorithm that schedules a group of several programs characterized by their time consuming very high on different supercomputers searching for an efficient assignment of the total running time. This efficient assignment grantees the fair load distribution of the execution on the supercomputers. The essential goal of this research is to propose several algorithms that can ensure the load balancing of the execution of all programs. In this research, all supercomputers are assumed to have the same hardware characteristics. The main objective is to minimize the gap between the total running time of the supercomputers. This minimization of the gap encompasses the development of novel solutions giving planning of the executable programs. Different algorithms are presented to minimize the gap in running time. The experimental study proves that the developed algorithms are efficient in terms of performance evaluation and running time. A comparison between the presented algorithms is discussed through different classes of instances where in total the number of instances reached 630. The experiments show that the efficient algorithm is the best-programs choice algorithm. Indeed, this algorithm reached the percentage of 72.86%, an average running time of 0.0121, and a gap value of 0.0545.

Citation: Algashami AM (2022) Algorithms for the executable programs planning on supercomputers. PLoS ONE 17(9): e0275099. https://doi.org/10.1371/journal.pone.0275099

Editor: Ali Safaa Sadiq, Nottingham Trent University School of Science and Technology, UNITED KINGDOM

Received: March 17, 2022

Accepted: September 11, 2022

Published: September 26, 2022

Data Availability Statement: All relevant data are within the manuscript.

Funding: The authors would like to thank the Deanship of Scientific Research at Majmaah University for supporting this work under project no R-2022-198.

Competing interests: The authors have declared that no competing interests exist.

1 Introduction

This paper focuses to develop an algorithm for the problem related to the scheduling of the execution programs by supercomputers. Different programs are received by the administrator to be scheduled on the available supercomputers. Each program has its own executing time. The supercomputers must be operated in the same way and with the same use of time. This is can be reached by an appropriate algorithm that can schedule the received programs with a fair way distribution. Otherwise, when fairness is not applicable, one supercomputer can be more exploited than another one.

The problem presented in this paper can be defined as follows. Suppose that there are several executable programs. These programs need a big amount of memory and a robust recourses like processor to ensure the accomplishment of the total execution. Each program is described by its own running time. The objective is concerned with finding a method that give a schedule for these executable programs guaranteeing a fair distribution in term of running

time. There is no previous research in the literature that studied the proposed problem. However, many research works can be referred to the load balancing problem.

The load balancing treated the budget distribution is developed in [1]. A mathematical formulation was proposed to give an objective function for the problem. This formulation applied the minimization of the gap in the cumulative income between different regions. Three algorithms were treated to solve the latter problem. The randomization procedure is used as the first approximate solution. The second one is an iterative algorithm. The last one is the moving of the probabilistic values. In the same context, another work treated the project assignment was studied in [2]. In this work, the authors formulated the problem by giving a new objective function. Indeed, a minimization of the maximum revenue was studied. In this latter research, different algorithms were developed. These algorithms utilize the dispatching rules method and the multi-fit method. Different groups of instances are tested to measure the efficacy of the given algorithms.

In [3], the authors developed solutions regarding the cloud repository problem using the load balancing procedures. This paper focus on the issues in relation to good operation of storage servers in the cloud environment. The main objective of this research will contribute in handling several challenges concerning the load balancing in the cloud environment. Though analyzing the researches discussing topics in this field, in this paper 2 distributed load-balancing procedures. Similar work can be cited as [4]. A review regarding the load-balancing approaches in cloud context is proposed in [5].

Load-balancing dispatches the workload through several nodes to obtain better results when exploited the system. Different load balancing procedures occur to reach better resource exploitation. In [6], authors developed a discussions of load balancing procedures. In addition, the authors in the latter paper, give a comparison between the proposed algorithms on the basis of different indicators like mean no-waiting time, processing time, and data cost time.

The load balancing procedures are used in literature in the domain of the projects and budgets distribution. Indeed, in [7], several lower bounds were proposed, different heuristics, and a exact method for the project distribution were dedicated to propose solution. In this latter paper, the objective function is proposed as a new one compared with the one given in [2]. In [1], the author proposed three heuristics to solve the problem of the projects revenues assignment problem. In the same context, in [8] the authors proposed an exact algorithm for the budget scheduling problem. Different heuristics and algorithms were proposed to be used in the exact approach. The load balancing problem is studied and applied on many domains in literature.

The load balancing used on the storage spaces is utilized in [9]. In fact, several algorithms were developed. These algorithms are used in different by the dispatching rules approach. A comparison between the proposed algorithms are discussed.

A load balancing procedures regarding the supercomputers are treated in different previous works. A periodic load balancing procedure are studied in [10]. The load balancing regarding the message-passing in supercomputer were treated in [11].

On the other hand, the utilization of the load balancing is adopted in the network field. Indeed, several algorithms were developed to find an acceptable solution that ensuring the equity transmission of the give data [12].

Another field that the load balancing is applied, is the aircraft field [13]. Authors proposed several lower bounds regarding the load balancing applied on the gas turbine field. These lower bounds are based basically on the iterative approach, subset sum problem and the knapsack problem. In [14], the authors treated the same problem studied in [13] by using the randomization method.

In the domain of health care, several clustering algorithms which are used two sets is the best algorithm with 96% for the small scale instances and 98% for the big instances. The novelty of this research is the utilization of the dispatching rules by different modifications; randomized method, clustering approach; probability application algorithm, and multi-start algorithm to schedule quality reports to available physicians. The objective is to guarantee the fair assignment of the number of papers workload [15].

Authors in [16] developed a cost model related to the correctness of the load imbalance. This model offers discussions of the efficiency of load balancing procedures in any particular imbalance case. The developed process, in this latter paper, correctly selects the algorithm that carry out the lowest running time in up to 96% of the total cases, and may accomplish a 19% profit over selecting a single balancing procedure for all generated cases.

Utilizing loop parallelism is obviously most critical in accomplishing high system and routine efficiency. Because of the clarity of this method, guided self-scheduling is specifically adapted for execution on real parallel machines. This approach accomplishes concurrently the two most significant goals: load balancing and extremely low synchronization overhead. For particular types of repeating the results prove analytically that guided self-scheduling utilize minimal overhead and accomplishes optimal schedules. Two other interesting properties of this approach are its insensitivity to the first processor configuration (in time) and its parameterized nature which enables us to tune it for different systems [17].

In the industrial domain, the load balancing algorithms are used in [18] to ensure a better utilization of the machines and guarantee an equity use of machines. In the same context, the authors in [19] solve the equity distribution of the jobs on the machines by the multi-start algorithms applying the probabilistic method.

A dynamic balancing algorithm assumes the low effects of two main elements of the system which related to job behavior and the general state of the system, i.e., load balancing procedure using the actual status of the system. The establishment of an efficacious dynamic load balancing procedure encompasses several essential issues: load cost, load standard comparison, assessment indicators, system stability, amount of data exchanged between nodes, job resource required, job's chosen for transfer, remote nodes chosen, and more [20].

In addition, dispatching the persons into vehicle ti ensure an equity distribution of these persons on the available parking is proposed in [21]. Recently, authors in [22] proposed novel algorithms to solve the parking managment. Several researches have been uploaded in the literature in order to cover some aspects in relation to the execution period and the gap valuation, which have been used to explore the assessment of the development procedures. Analyzing the gathered experiments shows a good indication in the assessment behavior of the proposed algorithm. In addition, it shows the proposed algorithm can narrow the gap in issues in relation to gap and time calculation in the developed researches. The *MR* heuristic reached an exceptional assessment results compared this result with the best algorithms proposed in [21]. The *MR* heuristic reached a percentage of 96%, a gap of 0.02, and a running time of 0.007s.

The usage of the supercomputers is largely referred in the literature. In fact, the utilization of the scheduling on the supercomputers filed may be cited to the different works. In [23], the authors studied the resource management through the usage of the supercomputers by an energy-performance procedure. The parallelism of the processing in the supercomputers is proposed by [24].

In [25], authors considered the methods of optimal load balancing and the storage requirements of algorithms. Other algorithms proposed in [26] can be utilized in future work to enhance the proposed algorithms. A mathematical model for scheduling activities while there are priorities between devices are proposed in [27]. In [28], authors treated the scheduling algorithms into networks. These algorithms can be exploited to give a new solutions for the

presented problem. In [29], authors developed algorithms for the category constraint into network based on scheduling problem. Authors in [30] developed algorithms for the read frequency of data. These algorithms may be exploited on and adopted for the presented problem. A recent similar work for the latter paper are proposed in [31].

In this paper, a mathematical model of the presented problem is proposed. In addition, many algorithms were proposed to manage the studied problem. The problem can be defined as follows. Several programs characterized by its estimated execution time need to be ran by several supercomputers. All the supercomputers are assumed to be characterized by the same criteria as the hardware. The goal is to search for an efficient algorithm to schedule these programs to the available supercomputers. This is can be mathematically written as the load balancing of programs to supercomputers.

The rest of the paper is structured as follows. Section 2 presents the justification and motivation to work the studied problem. In section 3, the problem definition is described. Section 4 presents the research method and design. The developed algorithms to solve the presented problem are detailed in Section 5. The experimental results and discussions are analysed in Section 6. Section 7 is reserved for the conclusions and perspectives.

2 Justification and motivation

The presented problem may be defined as follows. Suppose that there are a set of several programs characterized by its estimated running time. This set of programs need to be executed by many available supercomputers. The set of programs is homogeneous. This is means that all the programs in the set have the same hardware characteristics. In addition, these programs are supposed to consume more resources in memory and time execution. Consequently, it is primordial to seek an efficient way to schedule the given programs on the available supercomputers. This may be mathematically written as the equity assignment of programs to the supercomputers. The goal of this paper is to concept and design an algorithm that may minimize the execution time gap between all supercomputers. The first phase toward achieving the goal of this paper is to formulate mathematically the proposed problem. After that, this mathematical formulation is utilized to develop different algorithms to solve the presented problem. This is constitute the second phase. A detailed discussions and explanation of these two phases are presented in this paper.

3 Problem definition

Table 1 gives an overview for all variable notations and definitions used in the paper.

Example 1 explains the presented problem using all above definitions.

Example 1 *Suppose that* $n_{su} = 2$ *and* $n_{pr} = 7$. *The* e_p *value for each program is illustrated in* Table 2.

Now, an algorithm is ran to assign the programs detailed in Table 2 *to the available supercomputers. This algorithm is the shortest execution time algorithm, the obtained schedule is illustrated in* Fig 1. *It is evident to see that programs* {1, 5, 6, 7} *are executed by* Su_1 *and programs* {2, 3, 4} *are executed by* Su_2.

Fig 1 *shows that* Su_1 *has a total execution time of 1264. Moreover,* Su_2 *has a total execution time of 893. Accordingly, the execution time gap between* Su_1 *and* Su_2 *is* $Rt_1 - Rt_{min} = 1264 - 893 = 371$. *The goal is to conecpt and design an algorithm that can reduce the returned gap between supercomputers. In fact, for Example 1 another schedule must be given with a better solution comparing with schedule 1. This is means that an algorithm giving a gap less than 371.*

Table 1. Variable notations and definitions.

Variable	Definition
Pr	Set of programs that will be executed by the different supercomputers
n_{pr}	Number of programs delivered by the administrator
Sr	Set of supercomputers
n_{su}	Number of supercomputers
p	The program index
Pr_p	The program number p
i	The supercomputer index
Su_i	The supercomputer number i
e_p	The estimated execution time for the program p
ct_p	Cumulative execution time when p is scheduled
Rt_i	The total execution time for each i after accomplishing the execution of all programs
Rt_{min}	$\min_{i=\{1,\cdots,n_s\}} Rt_i$
A_b	The minimum gap value reached after finishing the workload of all algorithms
A	The gap value given by the presented algorithm
$Gp = \frac{A-A_b}{A}$	The gap between the minimum value and the presented one
$Time$	Average running time in seconds. In Tables "*" means that the execution time is less than 0.0001 s
Pcg	Percentage of programs where $A_b = A$ among all tested instances

https://doi.org/10.1371/journal.pone.0275099.t001

Table 2. The estimated running time for each program.

Pr_p	Pr_1	Pr_2	Pr_3	Pr_4	Pr_5	Pr_6	Pr_7
e_p	261	291	231	371	491	201	311

https://doi.org/10.1371/journal.pone.0275099.t002

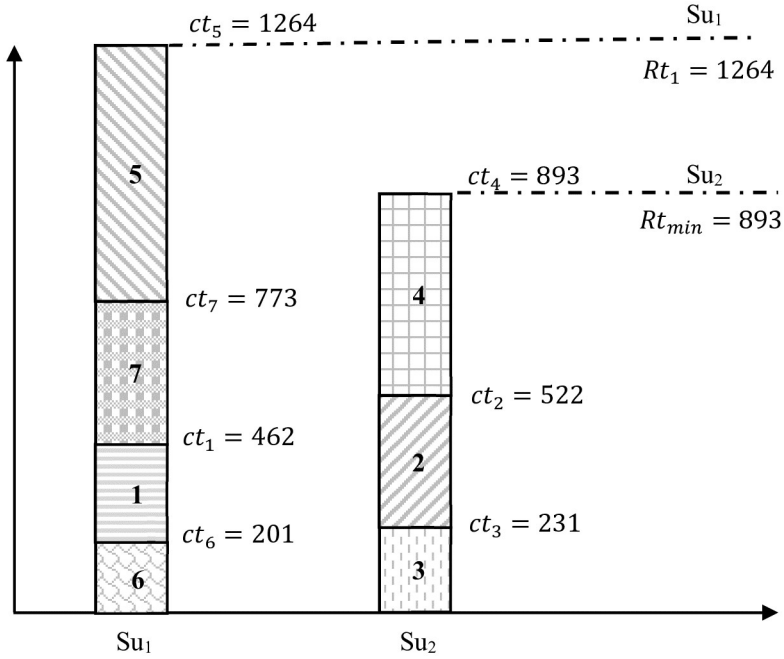

Fig 1. Shortest execution time schedule for Example 1.

https://doi.org/10.1371/journal.pone.0275099.g001

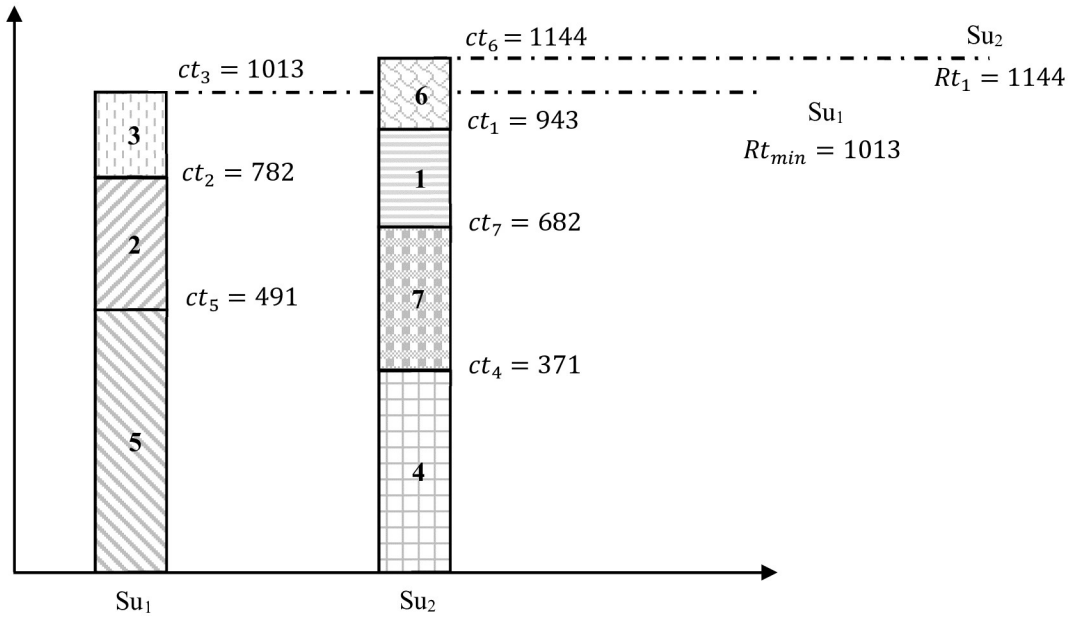

Fig 2. Longest execution time schedule for Example 2.

https://doi.org/10.1371/journal.pone.0275099.g002

Example 2 *For this example, the instance detailed in Table 2 is examined. Calling the longest execution time algorithm, the result given the schedule is illustrated in Fig 2. It is easy to see that programs* {2, 3, 5} *are executed by* Su_1 *and programs* {1, 4, 6, 7} *are executed by* Su_2.

Fig 2 shows that Su_1 *has a total execution time of 1013. Moreover,* Su_2 *has a total execution time of 1144. Thus, the execution time gap for* Su_1 *and* Su_2 *is* $Rt_2 - Rt_{min} = 1144 - 1013 = 131$. *For this example, the schedule 2 gives a better results than schedule 1.*

In general, facing on different supercomputers, an indicator must be determined to evaluate the gap of the algorithm that searching the load balancing. Eq 1 represent the gap of the execution. This gap is calculated between the supercomputer that having the minimum total execution time and all others supercomputers. This gap must be minimized to guarantee the load balancing. Hereafter, this gap is denoted by *Grt*.

$$Grt = \sum_{i=1}^{n_s} (Rt_i - Rt_{min}). \quad (1)$$

Proposition 1 *Based on Eq 1, the gap Grt may be formulated such that in Eq 2.*

$$Grt = \sum_{i=1}^{n_s} Rt_i - n_s Rt_{min}. \quad (2)$$

Proof 1 $Grt = \sum_{i=1}^{n_s}(Rt_i - Rt_{min}) = \sum_{i=1}^{n_s} Rt_i - \sum_{i=1}^{n_s} Rt_{min}$. *It is clear to see that* $\sum_{i=1}^{n_s} Rt_{min} = n_s Rt_{min}$. *Thus, the Eq 2 is obtained.*

4 Research method and design

In this section, six components are proposed for the proposed model. To more understand the model proposed in this research, an example is given of two supercomputers and four programs as shown in Fig 3. The first component is "supercomputer". For the example given in This component contain the supercomputer. The "data center" is a component that can

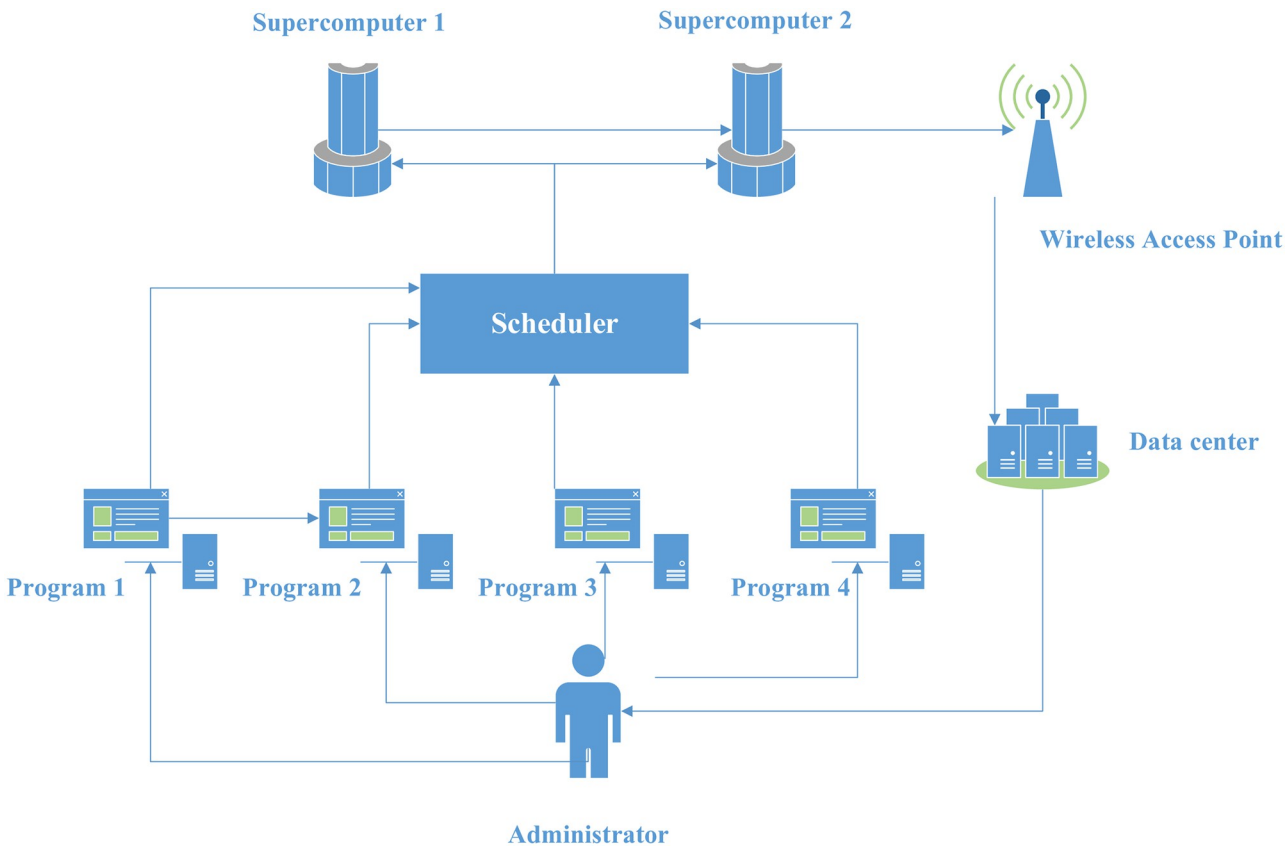

Fig 3. 4-programs and 2-supercomputers example for the design model.

https://doi.org/10.1371/journal.pone.0275099.g003

receive the results of the executed programs by the supercomputers. In addition, this component is responsible to send all new programs to be executed to the administrator. The component which guarantee an efficient transmission between the supercomputers and the data center is the "Wireless Access Point". The "administrator" is the component represented by the user that having all access authorization and can decide for choosing the programs sent by the data center. Finally, the component "scheduler" is responsible to apply the developed algorithms to propose a solution regarding the good distribution of programs to the different supercomputers. The main component in this research work is the "scheduler".

In this paper, a new design of the studied problem regarding the planning of the executable program on the available supercomputers is proposed as illustrated in Fig 3. The proposed components are focalized on the scheduler one. The scheduler is responsible to call all the algorithms to solve the scheduling problem and decide which program must be executed on the fixed supercomputer. Several variants of probabilistic method are proposed. In general, using a probabilistic method and the randomization approach give an efficient approximate solution for the scheduling solution problem. It is important to notice that the developed problem is NP-hard one. This is confirm that a good approximate solution represent a great archive for the studied problem.

Based on the example of 4-programs and 2-supercomputers shown in Fig 3, a generalization of the model as shown in Fig 4 can be illustrated. The general model is composed by five fundamental components: Supercomputers engine, scheduler, Programs engine, wireless access point, and data center.

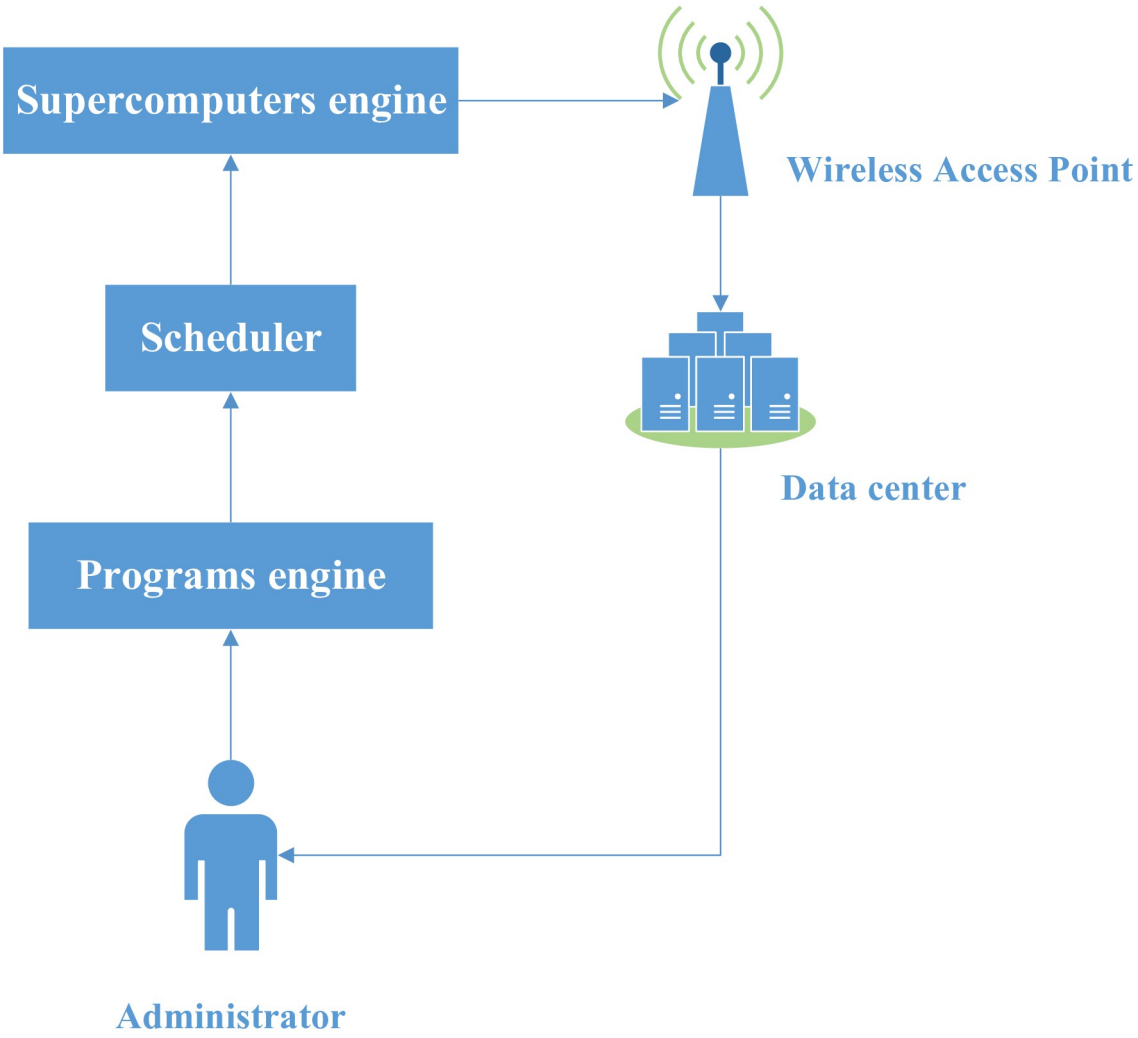

Fig 4. Design of the supercomputers-program model.

https://doi.org/10.1371/journal.pone.0275099.g004

5 Developed algorithms

In this section, seven proposed algorithms will be presented and detailed. Indeed, each algorithm will be explained and a pseudo-code will be illustrated for some algorithms to explain the functionality of these algorithms. The complexity of each algorithm is given.

5.1 Longest execution time algorithm

A dispatching rule method is applied for the longest-running time algorithm (*LTA*). Firstly, all programs are listed in the non-increasing order of its execution time. Next, the programs that have the longest execution time will be executed by the supercomputer that has the minimum ct_p and so on until completing all the workload.

5.2 Shortest execution time algorithm

A dispatching rule method is applied for the longest-running time algorithm (*STA*). Firstly, all programs are listed in the increasing order of its execution time. Next, the programs that have

the longest execution time will be scheduled on the supercomputer that has the minimum ct_p, and so on until completing all workload.

5.3 Programs choosing probabilistic algorithm (*PCP*)

This algorithm applies the probabilistic approach. The first program which will be chosen to be ran by the supercomputer is picked by a probability β. After that, the second program will be picked among the remaining programs applying the same probability β and so on until completing all workload. In the implementation code, the calculation of the probability is determined as follows. In the first, a number h is generated and returned randomly in $[1, n_s]$. Now, Pr_h is selected and executed by the available supercomputer. The set Pr will be updated by removing Pr_h from this set. So, $Pr = Pr \setminus Pr_h$. This procedure is repeated $Lmt = 1000$ times and the minimum value of the returned gap Grt will be stored.

Hereafter, $Rad(x, y)$ denoted the function that give a random number in $[x, y]$. $SLg(r, z)$ is the function that schedules Pr_r on Su_z.

This algorithm is *PCP* and all details are illustrated in Algorithm 1. The complexity of *PCP* algorithm is $O(n^2)$.

Algorithm 1 *PCP* algorithm

```
for (v = 1 to Lmt) do
  Fix x = n_su
  while (x ≥ 1) do
    Fix k = Rad(1, x)
    Available supercomputer is Su_z
    Call SLg(k, Su_z)
    Fix x- -
  end while
  Determine Grt_v
end for
Determine Grt = min_{1≤v≤Lmt} Grt_v
```

5.4 Non-decreasing-order-programs choosing probability algorithm

This algorithm applies the probability-approach as detailed in the Subsection 5.3. In the first, the programs is sorted in the non-decreasing order of its estimated execution time. The selected program that can be ran by the free supercomputer is picked by a probability γ. This instruction will be repeated $Lmt = 1000$ times and the minimum obtained gap Grt will be stored. Hereafter, $ICG(m)$ represent the function that sort Pr_m in the non-decreasing order of its estimated execution time. The complexity of *NCP* algorithm is $O(n^2)$.

This algorithm is denoted by *NCP*. The instructions of *NCP* are illustrated in Algorithm 2.

Algorithm 2 *NCP* algorithm

```
Call ICG(Pr)
for (v = 1 to Lmt) do
  Fix x = n_su
  while (x > 0) do
    Fix k = Rad(1, x)
    Available supercomputer is Su_z
    Call SLg(k, Su_z)
    Fix x- -
  end while
   Determine Grt_v
end for
Determine Grt = min_{1≤v≤Lmt} Grt_v
```

5.5 Decreasing-order-choosing probabilistic algorithm

This algorithm applies the randomization approach as detailed in the Subsection 5.3. In the first, the programs is sorted in the non-increasing order of its e_p. The picked program which will be ran by the free supercomputer is chosen by a probability γ. This instruction is repeated $Lmt = 1000$ times. The minimum gap will be stored. Denoted by $DRG(Pm)$ the procedure that sort the programs Pm in the non-increasing order of its estimated execution time. The complexity of DCP algorithm is $O(n^2)$.

This algorithm is *DCP*. The details of *DCP* are illustrated in Algorithm 3.

Algorithm 3 *DCP* algorithm

```
Fix DRG(Pr)
for (v = 1 to Lmt) do
  Fix x = n_su
  while (x > 0) do
    Fix k = Rad(1, x)
    Available supercomputer is Su_z
    Call SLg(k, Su_z)
    Fix x- -
  end while
  Determine Grt_v
end for
Determine Grt = min_{1≤v≤Lmt} Grt_v
Return Grt
```

4.6 Three-variant-programs choosing probabilistic algorithm

This algorithm applies the probabilistic approach as detailed in the above subsections. In first, *PCP* algorithm is called. The solution obtained by *PCP* is Denoted by *Grt*1. After that, the *NCP* algorithm is called. Denoted by *Grt*2 the returned solution. Finally, *DSP* algorithm is called and denoted by *Grt*3 the returned solution. The best solution between *Grt*1, *Grt*2 and *Grt*3 is picked. The complexity of *TSP* is $O(n^2)$. Denoted by $PCP(Pm)$, $NCP(Pm)$, and $DCP(Pm)$ the functions calling the algorithms *PCP*, *NCP*, and *DCP*, respectively.

This algorithm is *TVP*. The details of *TVP* are illustrated in Algorithm 4.

Algorithm 4 *TVP* algorithm

```
Call PCP(Pm)
Determine Grt1.
Call NCP(Pr)
Determine Grt2.
Call DCP(Pr)
Determine Grt3.
Determine Grt = min(Grt1, Grt2, Grt3)
Return Grt
```

5.7 Best-programs choosing algorithm

This algorithm use *LRT* and *TVP* algorithms. Indeed, *LRT* and *TVP* algorithms are called separately and the best result is selected. This algorithm is *BPC*.

6 Experimental results and discussions

Many indicators are given to assess the efficiency of the presented algorithms. Through these indicators, a comparison between the proposed algorithms are discussed. All proposed algorithms were implemented in C++. The computer executing all the developed code is an Intel (R) Core (TM) i5-3337U CPU and the operating system is Windows 10.

6.1 Instances and tests

Three classes are proposed in the subsection "Instances" to measure the efficiency of the presented algorithms and three indicators are presented in the subsection "Tests".

6.1.1 Instances. Different instances are tested and experimented. The types of classes applied in this paper are the uniform distribution which is denoted by $UN[x_1, x_2]$. The comparative study between the algorithms and the manner that the instances are generated are inspired from the study [32].

The e_p values will be as:

- Class A: $x_1 = 1$ and $x_2 = 100$, $e_p \in UN[1, 100]$.
- Class B: $x_1 = 10$ and $x_2 = 150$, $e_p \in UN[10, 150]$.
- Class C: $x_1 = 100$ and $x_2 = 500$, $e_p \in UN[100, 500]$.

The permutation of the pair (n_{pr}, n_{su}) discussed in this experimental results are listed as follows. The small scale is for $n_{pr} = \{10, 25, 30\}$ the number of supercomputers is $n_{su} = \{4, 5, 6\}$. The big scale is for $n_{pr} = \{50, 60, 80, 100\}$ the number of supercomputers is $n_{su} = \{5, 6, 10, 12\}$

For each pair (n_{pr}, n_{su}) and each class, 10 instances were tested. The total generated instances is 630.

6.1.2 Tests. The indicators G_b, *Time*, and *Per* used to asses the algorithms are defined in Table 1.

Fig 5 represented the performance test measurement of the proposed algorithms. In this latter figure, it is supposed that there are four different values of the *Grt* obtained by four different algorithms. These values are Grt_1, Grt_2, Grt_3 and Grt_4. It is clear to see that Grt_1 is better than Grt_2, Grt_3 and Grt_4 because Grt_1 is the minimum value and the objective is to minimize *Grt*. The value *LB* in Fig 5 represented the value of the lower bound for the studied problem. In general, the exact solution is in *LB*, *Grt*, with *Grt* is the value obtained by any algorithm. The closest value of *Grt* to *LB* is reached by Grt_1. This is meaning the interval $[LB, Grt_1]$ which represent the exact solution interval is the smallest interval comparing when choosing Grt_2, Grt_3 and Grt_4. This is prove that the choice of G_b value to asses the performance of the developed algorithms.

6.2 Results

Table 3 illustrated the overview of all algorithms. The variation of *Per*, G_b, and *Time* are presented in this latter table. The best algorithm that have the minimum gap is *BPC* reaching a percentage of 72.86%, an average gap of 0.0545 and average execution time of 0.0121 s. The second best algorithm is *TVP* reaching a *Pcg* value of 45.08%, a *Gp* value of 0.1944, and a *Time* value of 0.0121 s. The *STA* never obtained a minimum gap value.

Table 4 shows the *Gp* values variation for all proposed algorithms according to the number of programs. This table displays that the best *Gp* value of < 0.0001 is reached by *TVP* and *BPC* where $n_{pr} = 10$. On other hand, a maximum *Gp* value of 0.8905 is reordered by *SRT* where $n_{pr} = 30$. For $n_{pr} = 100$, the best *Gp* value of 0.0531 is recorded by *BPC* and a maximum *Gp* value of 0.7803 is recorded by *STA*.

Table 5 shows the *Gp* values for the proposed algorithms when n_{su} according to the number of supercomputers. This table displays that the best *Gp* value of 0.0263 is reached by *BPC* where $n_{su} = 14$. On other hand, a maximum *Gp* value of 0.8817 is returned by *SRT* where $n_{su} = 4$. Where $n_{su} = 14$, a maximum *Gp* value of 0.6952 is returned by *STA*.

Table 6 illustrates the *Gp* values for all algorithms and for each class.

Table 7 shows the *Time* variation for all proposed algorithms when n_{pr} changes. For *BPC* algorithm, the *Time* values increases when n_{pr} increase. In addition, the minimum *Time* value

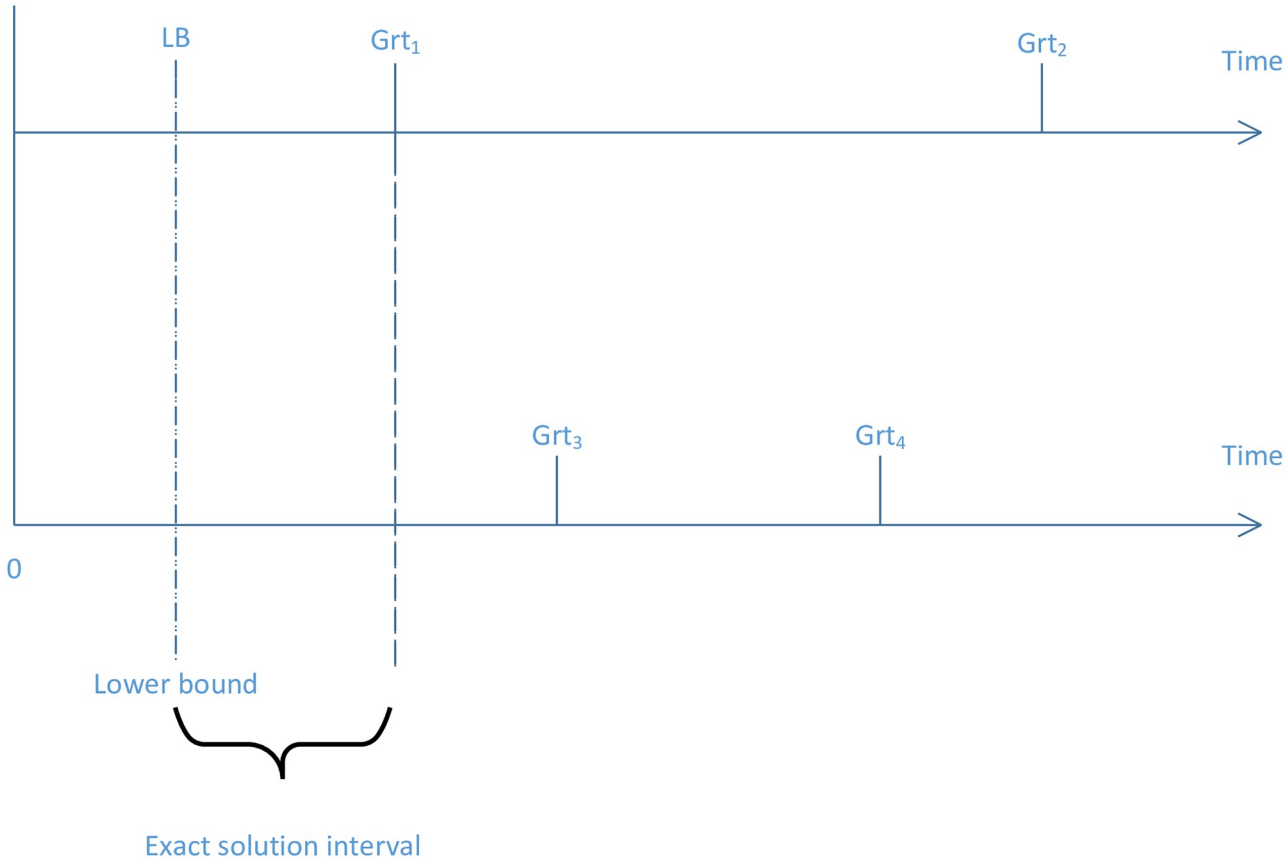

Fig 5. Performance test measurement.

https://doi.org/10.1371/journal.pone.0275099.g005

Table 3. Overview of all proposed algorithms.

	LTA	*STA*	*PCP*	*NCP*	*DCP*	*TVP*	*BPC*
Pcg	38.10%	0.00%	25.87%	25.40%	25.56%	45.08%	72.86%
Gp	0.3508	0.7626	0.2877	0.2924	0.2937	0.1944	0.0545
Time	*	*	0.0041	0.0042	0.0041	0.0121	0.0121

https://doi.org/10.1371/journal.pone.0275099.t003

Table 4. The *Gp* values variation for all proposed algorithms according to the number of programs.

n_{pr}	*LTA*	*STA*	*PCP*	*NCP*	*DCP*	*TVP*	*BPC*
10	0.1954	0.4983	0.0016	0.0103	0.0117	0.0000	0.0000
25	0.4607	0.8321	0.3165	0.3183	0.2746	0.1212	0.0836
30	0.5422	0.8905	0.3840	0.3897	0.4094	0.2876	0.1072
50	0.3501	0.7428	0.2369	0.2648	0.2638	0.1337	0.0557
60	0.0181	0.8118	0.5773	0.5610	0.5652	0.5141	0.0050
80	0.5652	0.7824	0.1805	0.1634	0.2055	0.0771	0.0771
100	0.3242	0.7803	0.3173	0.3392	0.3254	0.2269	0.0531

https://doi.org/10.1371/journal.pone.0275099.t004

Table 5. The *Gp* values for all algorithms when n_{su} according to the number of supercomputers.

n_{su}	*LTA*	*STA*	*PCP*	*NCP*	*DCP*	*TVP*	*BPC*
4	0.7875	0.8817	0.2524	0.2733	0.2631	0.1240	0.1240
6	0.4217	0.8021	0.2921	0.3103	0.3137	0.1773	0.0673
8	0.2625	0.6156	0.1363	0.1510	0.1413	0.0784	0.0413
12	0.1791	0.7549	0.3624	0.3488	0.3612	0.2895	0.0305
14	0.2374	0.6952	0.2501	0.2467	0.2424	0.1735	0.0263

https://doi.org/10.1371/journal.pone.0275099.t005

Table 6. The *Gp* values for all algorithms and for each class.

Class	*LTA*	*STA*	*PCP*	*NCP*	*DCP*	*TVP*	*BPC*
1	0.3582	0.7828	0.3010	0.3072	0.2875	0.1989	0.0545
2	0.3689	0.7014	0.2679	0.2764	0.2802	0.1927	0.0586
3	0.3254	0.8035	0.2943	0.2935	0.3132	0.1915	0.0506

https://doi.org/10.1371/journal.pone.0275099.t006

Table 7. The *Time* variation for all proposed algorithms when n_{pr} changes.

n_{pr}	*LTA*	*STA*	*PCP*	*NCP*	*DCP*	*TVP*	*BPC*
10	*	*	0.0008	0.0012	0.0010	0.0027	0.0027
25	*	*	0.0018	0.0020	0.0020	0.0058	0.0058
30	*	*	0.0024	0.0024	0.0022	0.0061	0.0061
50	*	*	0.0045	0.0044	0.0046	0.0123	0.0123
60	*	*	0.0049	0.0048	0.0048	0.0150	0.0150
80	*	*	0.0066	0.0065	0.0063	0.0195	0.0196
100	*	*	0.0079	0.0083	0.0076	0.0236	0.0236

https://doi.org/10.1371/journal.pone.0275099.t007

Table 8. The *Time* variation for all proposed algorithms when n_{su} changes.

n_{su}	*LTA*	*STA*	*PCP*	*NCP*	*DCP*	*TVP*	*BPC*
4	*	*	0.0014	0.0016	0.0016	0.0042	0.0042
6	*	*	0.0038	0.0038	0.0039	0.0108	0.0108
8	*	*	0.0021	0.0023	0.0020	0.0052	0.0052
12	*	*	0.0063	0.0065	0.0060	0.0192	0.0192
14	*	*	0.0065	0.0067	0.0064	0.0198	0.0198

https://doi.org/10.1371/journal.pone.0275099.t008

of 0.0027 s is returned where n_{pr} = 10 and the maximum *Time* value of 0.0236 s is returned where n_{pr} = 100.

Table 8 shows the *Time* variation for all proposed algorithms when n_{su} changes.

Each triple (n_{pr}, n_{su}, *class*) will be denotes by *Tp*. The values of n_{pr} are {10, 25, 30, 50, 60, 80, 100} and the values of n_{su} are {4, 6, 8, 12, 14}. Three classes are proposed. So, in total 63 values of *Tp* are presented. Fig 6 shows the *Gp* values variation when *Tp* changes for *BPC*.

Fig 7 shows the *Time* behavior when *Tp* changes for *BPC*. This figure prove that the *Time* values are constantly increasing when n_{pr} increase. The maximum *Time* value of 0.0281 s is returned where (n_{pr}, n_{su}, *class*) = (100, 14, 1) and the minimum *Time* value of 0.0022 s is returned where (n_{pr}, n_{su}, *class*) = (10, 4, 1).

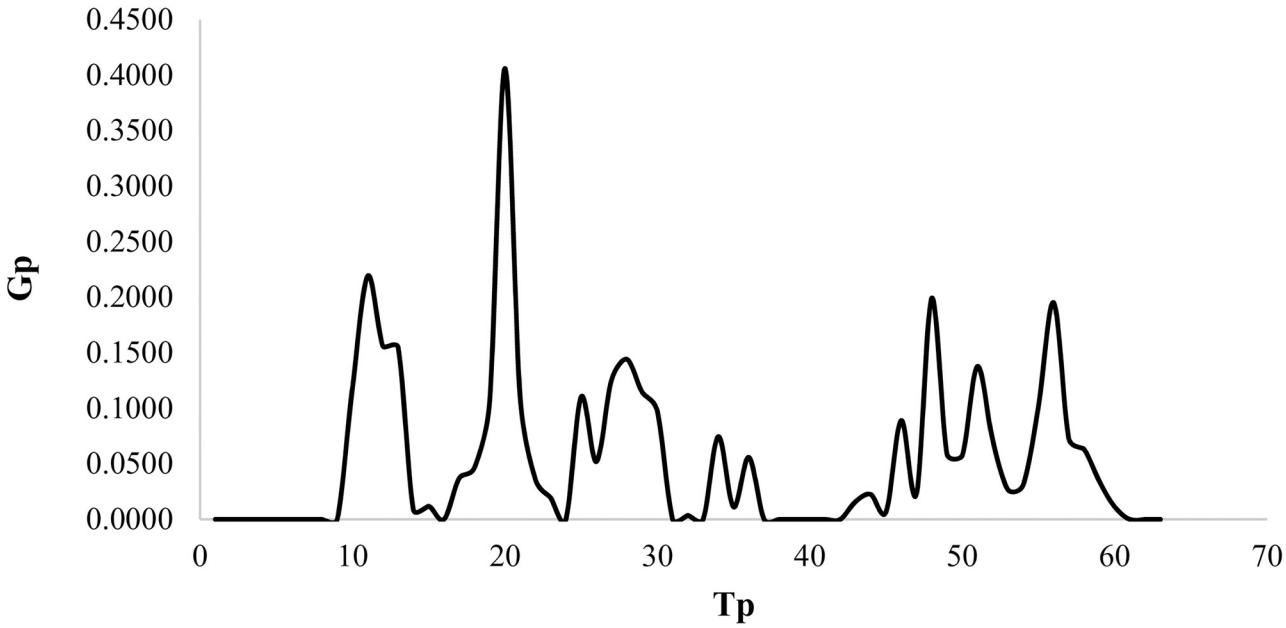

Fig 6. The *Gp* values variation when *Tp* changes for algorithm *BPC*.

https://doi.org/10.1371/journal.pone.0275099.g006

6.3 Discussions

Seven algorithms are proposed and tested in this paper. Table 3 shows the overview of the results given by all algorithms. this table shows that the best algorithm that reached the maximum percentage is *BPC* with a percentage of 72.86%. This percentage is not reaching the value of 100%, this is means that there is no dominance between the proposed algorithms. Indeed,

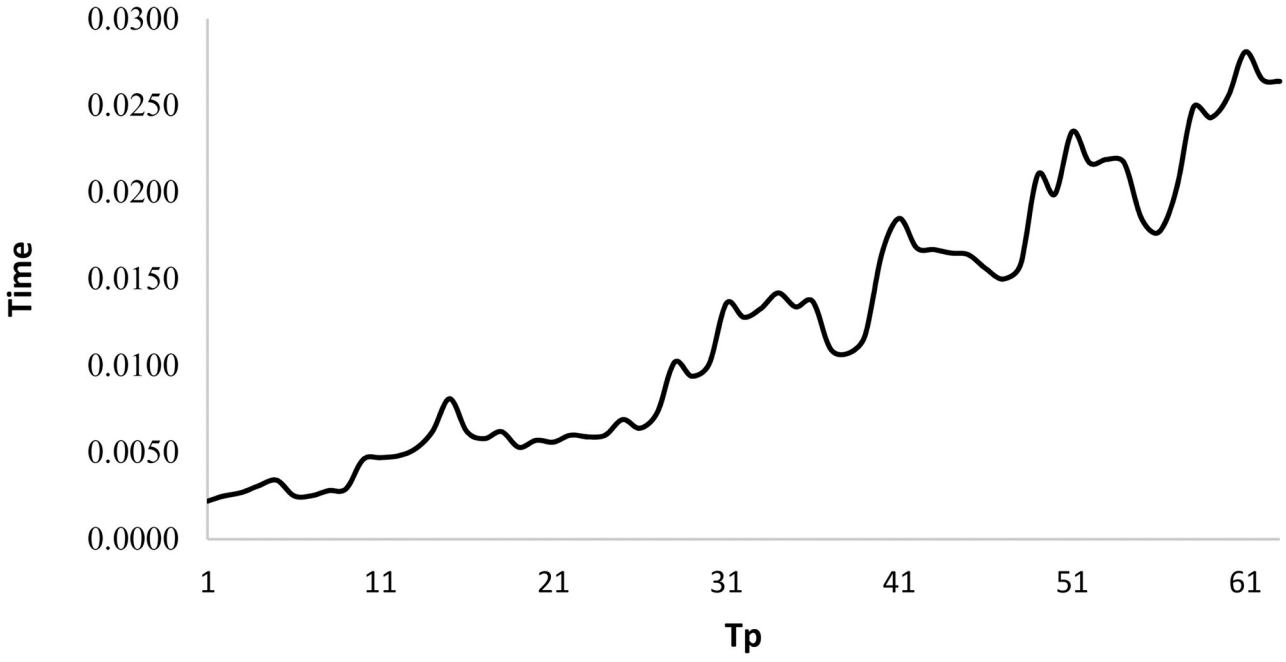

Fig 7. Time variation when *Tp* changes for algorithm *BPC*.

https://doi.org/10.1371/journal.pone.0275099.g007

for certain instances, the other algorithms excepting *BPC* give best results than *BPC* but in total *BPC* reaches 72.86%. The algorithms *PCP*, *NCP*, and *DCP* give closer results in the range of 25%. The maximum average time obtained by all algorithms is around 0.0121 s. This gives a remarkable runtime to reach an efficient solution. It is worth noting that, the *TVP* reached a 45.08% as performance compared with *BPC* that has 72.86% as a performance, in the same average runtime of 0.0121 s. Therefore, by consuming the same time a better solution can be reached by choosing the algorithm *BPC*. In this paper, the choice of three classes is based on several works in literature that use these classes and the generation of the instance to test the obtained results. Indeed, the choice is to give a variety of different ranges of the scale of the proposed problem. For Class 1, the range is for the small instances. For Class 2, the range is for the medium instances. Finally, Class 3 is for the big-scale instances reaching 500 programs. On other hand, a different range of scales is proposed for the number of programs and the number of supercomputers. The runtime of the algorithms is related to the structure of the utilization of the loop instructions in each algorithm. For *LTA* and *STA*, the average execution time is always less than 0.0001 s. This is obtained because these algorithms utilizing the dispatching-rules which are executed by calling the heap-sort algorithm which is $O(nlogn)$. However, for algorithms *PCP*, *NCP* and *DCP* the complexity is $O(n^2)$. This explains the average execution time which is very close.

The algorithms proposed in this paper are based on several variants of the probabilistic method. In practice, the probabilistic method and the randomization approach provide a better solution for the scheduling problem. This is due to the multiple choice selection of the programs and the repetitive procedure. This observation is always applicable for the studied problem when the best-programs choosing algorithm is the best compared with other algorithms. Consider the two case studies for the presented problem: the small-scale instances and the big-scale instances. The small-scale instances are the instances such as $n_{pr} \leq 50$. However, the big-scale instances are the instances such as $n_{pr} > 50$. The *BPC* algorithm gives remarkable results for the first case study which is the small-scale instances and for the second case study which is the big-scale instances in term of gap. Indeed, the average gap of all small-scale instances is 0.0616 and the average gap of all big-scale instances is 0.0451. In term of running time, the *BPC* algorithm gives remarkable results for the first case study and for the second case study. Indeed, the average running time of all small-scale instances is 0.0067 and the average running time of all big-scale instances is 0.0194. The proposed algorithms can be applied on several real applications. Indeed, the parallel computing which is a type of computation in which several calculations must be executed at the same time. Big problems can be classed into many small problems, which may then be solved simultaneously. Other real application of the proposed algorithms is the parallel programming which is the process of utilizing a group of resources to solve a problem in minimum time by partitioning the big work. It is worth noting that the studied problem is an NP-hard one. Consequently, a good approximate solution represents a great archive for the studied problem. Several meta-heuristics can be applied using the proposed algorithms to enhance the obtained results.

7 Conclusion

This paper discussed the program's planning problem. These programs must be executed by a fixed number of available supercomputers. Each program requires a long period of time to be executed. the presented problem is strongly very hard. Especially, the hardness of the problem appears face on a big number of programs. In this paper, seven algorithms were developed to present solution of the studied problem. These algorithms utilize the dispatching rules and the randomization approach with several variants. The experimental results show that an efficient

feasible solution can be offered in an acceptable running time. The best-developed algorithm is the best-programs choosing algorithm in 72.86% of instance cases. In addition, the experiments show that the developed algorithms do not impose any dominance on them.

From a perspective of the studied problem, the developed algorithms can be exploited to develop a new better solution based on metaheuristics like a genetic algorithm or swarm optimization. The optimal of the studied problem can be constructed by an exact solution method based on the tree searching. In this case, the developed algorithms will be used to test and calculate certain rules in the node of the research tree.

Author Contributions

Conceptualization: Abdullah M. Algashami.

Formal analysis: Abdullah M. Algashami.

Methodology: Abdullah M. Algashami.

Supervision: Abdullah M. Algashami.

Validation: Abdullah M. Algashami.

Visualization: Abdullah M. Algashami.

Writing – original draft: Abdullah M. Algashami.

Writing – review & editing: Abdullah M. Algashami.

References

1. Jemmali M. Approximate solutions for the projects revenues assignment problem. Communications in Mathematics and Applications. 2019; 10(3):653–658. https://doi.org/10.26713/cma.v10i3.1238
2. Jemmali M. Budgets balancing algorithms for the projects assignment. International Journal of Advanced Computer Science and Applications. 2019; 10(11):574–578. https://doi.org/10.14569/IJACSA.2019.0101177
3. Gupta Y. Novel distributed load balancing algorithms in cloud storage. Expert Systems with Applications. 2021; 186:115713. https://doi.org/10.1016/j.eswa.2021.115713
4. Agarwal R, Sharma DK. Machine learning & Deep learning based Load Balancing Algorithms techniques in Cloud Computing. In: 2021 International Conference on Innovative Practices in Technology and Management (ICIPTM). IEEE; 2021. p. 249–254.
5. Shafiq DA, Jhanjhi N, Abdullah A. Load balancing techniques in cloud computing environment: A review. Journal of King Saud University-Computer and Information Sciences. 2021.
6. Nandal P, Bura D, Singh M, Kumar S. Analysis of Different Load Balancing Algorithms in Cloud Computing. International Journal of Cloud Applications and Computing (IJCAC). 2021; 11(4):100–112. https://doi.org/10.4018/IJCAC.2021100106
7. Alharbi M, Jemmali M. Algorithms for investment project distribution on regions. Computational Intelligence and Neuroscience. 2020; 2020. https://doi.org/10.1155/2020/3607547 PMID: 32802026
8. Jemmali M. An optimal solution for the budgets assignment problem. RAIRO–Operations Research. 2021; 55(2).
9. Alquhayz H, Jemmali M, Otoom MM. Dispatching-rule variants algorithms for used spaces of storage supports. Discrete Dynamics in Nature and Society. 2020; 2020. https://doi.org/10.1155/2020/1072485
10. Zheng G, Bhatele A, Meneses E, Kale LV. Periodic hierarchical load balancing for large supercomputers. The International Journal of High Performance Computing Applications. 2011; 25(4):371–385. https://doi.org/10.1177/1094342010394383
11. Xu J, Hwang K. Heuristic methods for dynamic load balancing in a message-passing supercomputer. In: Conference on High Performance Networking and Computing: Proceedings of the 1990 ACM/IEEE conference on Supercomputing. vol. 12; 1990. p. 888–897.
12. Jemmali M, Alquhayz H. Equity data distribution algorithms on identical routers. In: International Conference on Innovative Computing and Communications. Springer; 2020. p. 297–305.

13. Jemmali M, Melhim LKB, Alharbi SOB, Bajahzar AS. Lower bounds for gas turbines aircraft engines. Communications in Mathematics and Applications. 2019; 10(3):637–642. https://doi.org/10.26713/cma.v10i3.1218

14. Jemmali M, Melhim LKB, Alharbi M. Randomized-variants lower bounds for gas turbines aircraft engines. In: World Congress on Global Optimization. Springer; 2019. p. 949–956.

15. Jemmali M, Melhim LKB, Alourani A, Alam MM. Equity distribution of quality evaluation reports to doctors in health care organizations. PeerJ Computer Science. 2022; 8:e819. https://doi.org/10.7717/peerj-cs.819 PMID: 35174262

16. Pearce O, Gamblin T, De Supinski BR, Schulz M, Amato NM. Quantifying the effectiveness of load balance algorithms. In: Proceedings of the 26th ACM international conference on Supercomputing; 2012. p. 185–194.

17. Polychronopoulos CD, Kuck DJ. Guided self-scheduling: A practical scheduling scheme for parallel supercomputers. Ieee transactions on computers. 1987; 100(12):1425–1439. https://doi.org/10.1109/TC.1987.5009495

18. Jemmali M, Alourani A. Mathematical model bounds for maximizing the minimum completion time problem. Journal of Applied Mathematics and Computational Mechanics. 2021; 20(4):43–50. https://doi.org/10.17512/jamcm.2021.4.04

19. Jemmali M, Otoom MM, al Fayez F. Max-min probabilistic algorithms for parallel machines. In: Proceedings of the 2020 International Conference on Industrial Engineering and Industrial Management; 2020. p. 19–24.

20. Alakeel AM, et al. A guide to dynamic load balancing in distributed computer systems. International Journal of Computer Science and Information Security. 2010; 10(6):153–160.

21. Jemmali M. Intelligent algorithms and complex system for a smart parking for vaccine delivery center of COVID-19. Complex & Intelligent Systems. 2021; p. 1–13.

22. Jemmali M, Melhim LKB, Alharbi MT, Bajahzar A, Omri MN. Smart-parking management algorithms in smart city. Scientific Reports. 2022; 12(1):1–15. https://doi.org/10.1038/s41598-022-10076-4 PMID: 35444220

23. Kiselev E, Telegin P, Shabanov B. An energy-efficient scheduling algorithm for shared facility supercomputer centers. Lobachevskii Journal of Mathematics. 2021; 42(11):2554–2561. https://doi.org/10.1134/S1995080221110147

24. Lin FPC, Phoa FKH. Runtime estimation and scheduling on parallel processing supercomputers via instance-based learning and swarm intelligence. International Journal of Machine Learning and Computing. 2019; 9(5). https://doi.org/10.18178/ijmlc.2019.9.5.845

25. Kameda H, Li J, Kim C, Zhang Y. Optimal load balancing in distributed computer systems. Springer Science & Business Media; 2012.

26. Delgoshaei A, Ariffin M, Baharudin B, Leman Z. Minimizing makespan of a resource-constrained scheduling problem: A hybrid greedy and genetic algorithms. International Journal of Industrial Engineering Computations. 2015; 6(4):503–520. https://doi.org/10.5267/j.ijiec.2015.5.002

27. Delgoshaei A, Rabczuk T, Ali A, Ariffin MKA. An applicable method for modifying over-allocated multi-mode resource constraint schedules in the presence of preemptive resources. Annals of Operations Research. 2017; 259(1):85–117. https://doi.org/10.1007/s10479-016-2336-8

28. Alquhayz H, Jemmali M. Fixed Urgent Window Pass for a Wireless Network with User Preferences. Wireless Personal Communications. 2021; 120(2):1565–1591. https://doi.org/10.1007/s11277-021-08524-x

29. Sarhan A, Jemmali M, Ben Hmida A. Two routers network architecture and scheduling algorithms under packet category classification constraint. In: The 5th International Conference on Future Networks & Distributed Systems; 2021. p. 119–127.

30. al Fayez F, Melhim LKB, Jemmali M. Heuristics to Optimize the Reading of Railway Sensors Data. In: 2019 6th International Conference on Control, Decision and Information Technologies (CoDIT). IEEE; 2019. p. 1676–1681.

31. Jemmali M, Melhim LKB, Al Fayez F. Real time read-frequency optimization for railway monitoring system. RAIRO-Operations Research. 2022; 56(4):2721–2749. https://doi.org/10.1051/ro/2022094

32. Ben Hmida A, Jemmali M. Near-Optimal Solutions for Mold Constraints on Two Parallel Machines. Studies in Informatics and Control. 2022; 31(1):71–78. https://doi.org/10.24846/v31i1y202207