International Journal of Computer (IJC)
ISSN 2307-4523 (Print & Online)
https://ijcjournal.org/index.php/InternationalJournalOfComputer/index

Makespan Minimization for Efficient Placement of Distributed Computations on Virtual Dynamic Environment

Albert Djakene Wandala[a]*, Omer Yenke Blaise[b]

[a,b]*University of Ngaoundere, , Cameroon*
[a]*Email: djakene.albert@univ-ndere.cm*
[b]*Email: boyenke@univ-ndere.cm*

Abstract

Nowadays, virtualization, containerization technology and computer development make it possible to build distributed systems with virtual nodes, offering considerable performance for the execution of distributed computations. However, building such infrastructure faces various challenges of distributed systems, including load balancing, fault tolerance and wise placement of distributed computations on compute nodes. In this paper, we focus on the efficient placement of distributed computations in a virtual distributed system with the aim of minimizing the makespan. Several approaches have been proposed to reduce the placement makespan , but the need of improvement still remains. Consequently, in this work, we propose a new approach that minimizes the makespan of distributed computations on compute nodes by performing fine-grained intelligent placement. The results obtained during tests have shown a better placement of distributed computations on core nodes than existing approaches, regardless of the characteristics of the processes, cores, distributed computations and compute nodes.

Keywords: Placement; Makespan; Distributed System; Distributed computations; Virtualization.

1. Introduction

With current computer performance and virtualization technologies, it is possible to increase the performance of distributed systems in performing distributed computations [1,2] using virtual machines and containers. Unfortunately, this does not overcome the various challenges, including task placement, load balancing and fault tolerance. Among these challenges, this article focuses on the placement of distributed computations in a virtual distributed system with the aim of minimizing makespan.

Received: 7/16/2024
Accepted: 9/16/2024
Published: 9/26/2024

---

* Corresponding author.

To achieve this, it is necessary to implement approaches for placing distributed computations on computing units in the distributed system in order to minimize the makespan. The proposed approach allows the placement of distributed computing processes on node cores, regardless of the characteristics of the processes, cores, distributed computations and nodes. This document is organized into four sections. The first two sections present a state of the art of existing approaches, generalities on distributed systems, distributed computing, virtualization and placement, focusing on the tools and techniques used. The third section is devoted to the proposal of our virtual infrastructure, which integrates the placement mechanisms of distributed computations. The paper ends with a conclusion and perspectives.

2. Background

2.1. Distributed System

A distributed system can be defined as a set of autonomous computer entities that can communicate and exchange information with each other in order to solve a global problem. To solve this global problem, distributed systems have evolved over time, as have the different types of programming [3,4,5]. This began with single-tasking distributed systems, then supercomputers, then multitasking distributed systems, and finally the use of virtual machines and networked containers to solve complex and diverse problems that are distributed computations. To perform these computations, distributed systems need to have mechanisms for placing tasks on computing units.

2.2. Virtualization

In a datacenter, virtualization is symbolized by the use of virtual machines and/or containers as the execution environment. A virtualization tool enables several operating systems to run on the same physical machine or on the same distributed system. There are several examples of hypervisors [6,7,8], such as Qemu-KVM (Quick EMUlator/Kernel-based Virtual Machine) and Docker. Qemu-KVM is a type I hypervisor that is integrated into the Linux kernel and is easy to implement. It was designed from one branch of QEMU, which in turn integrates the source code of the other, so that the two are interdependent. Qemu-KVM is capable of executing machine code directly on the host processor to speed up emulation. Docker is a hypervisor for running computations in software containers. Docker enables the implementation of containers operating in isolation, via a high-level API. It relies instead on kernel functionality and uses resource isolation and distinct namespaces to isolate the operating system as it is perceived by the compute.

2.3. Placement of tasks

The authors of [9,10,13] define placement like an operation which consists of assigning elementary tasks to computing units. In a distributed system, placement consists of allocating the processes of one or more distributed computations to the processors of one or more computations nodes. The process of placement faces a number of challenges, including resource availability, dependency between tasks, and minimizing response time during communications. These different challenges make placement an NP-complete problem [11,12] that is widely treated in the literature. Thus, it is necessary to place computations on nodes taking into account a certain

number of criteria. In fact, several heuristics and algorithms have been developed in order to approach an optimal allocation in different problem cases. These algorithms can be divided into two main types: static placement and dynamic placement.

2.3.1. Static placement

The aim of static placement is to reduce the execution time of an application by assigning tasks to the available processors (or machines). This allocation is calculated at compile time and requires knowledge of all the tasks, their precedence relationships and their communication costs. There are two use cases for this type of placement:

- Case of independent distributed computations: In this case, it is sufficient to place the tasks of the computations according to the load on the resources. Several heuristics have been proposed in the literature, including MET (Minimum Execution Time), MCT (Minimum Completion Time), Min-Min, Max-Min and genetic algorithms.
- Case of dependent distributed computations: In this case, the system must place tasks according to the resource load and the dependencies between tasks. The heuristics developed to deal with this include list heuristics, task grouping heuristics, task duplication heuristics and meta-heuristics.

2.3.2. Dynamic placement

To compensate for the lack of static placement algorithms, researchers have experimented with several types of dynamic placement. Since choosing the best node is more or less easy, most of the work focuses on optimizing communications between nodes, in particular to avoid network congestion. There are two types of dynamic placement:

- Case of independent distributed computations: Several heuristics exist in this case, including the duplication heuristic, the WQR (Workqueue with Replication) heuristic and the approach of Bansal and his colleagues.
- Case of dependent distributed computations: In this case, the objective approach of Maria Chtepen and his colleagues is to maximize the number of jobs processed in a particular time interval and the approach of Gomathi and his colleagues who present a task scheduling model based on the grouping of tasks and makes it possible to maximize the use of resources and minimize task processing times.

3. Related Works

Several works in the literature have attempted to place tasks on compute nodes using several approaches. Firstly, the classic Min-Min approach, found in several works [14,15], which consists of placing tasks according to the minimum execution time. However, this approach considers tasks not placed at each decision and is not suitable for task balancing. And also, the classic Max-Min approach which, on the contrary, consists of placing tasks according to the maximum execution time and which has the disadvantage of increasing the overall execution time [16,15]. Singh Lalla and his colleagues [15] proposed in 2015 an approach called Improved-Min-

Min which aims to improve the classical Min-Min approach by improving the load balance and Makespan. In 2017, Konjaang James and his colleagues [17] proposed an approach called Improved-Max-Min which consists of improving the classic Max-Min by minimizing the overall execution time. In 2023, Sallar Sallam Murad and his colleagues [18] proposed an approach called Improved-Min-Min, the aim of which is to overcome the shortcomings of Min-Min by improving the Makespan and optimizing the use of resources, regardless of the size of the infrastructure.

However, these approaches work on the principle of placing a task on a node, whether or not the node is multi-core or whether or not the task is a distributed computation. The consequence of these approaches is its incapacity to take full advantage of the capabilities offered by the compute nodes, its incapacity to optimize the use of resources and therefore gives a more considerable Makespan. Given these limitations, the challenge of our approach is to minimize the Makespan by improving the use of resources. This improvement consists of placing each process of each task on a core of a compute node that takes into account the performance of each node core and the loads of each process of each task. This involves exploring the loads of the tasks, the loads of their processes, the performance of the nodes and the performance of their different cores, with the aim of making a more targeted and more powerful placement.

4. Structure of the proposed virtual architecture, Problem description and Methodology

4.1. Structure of the proposed virtual architecture

As part of this work, the robust distributed computing infrastructure is made up of mainly of physical machines (PM), virtual machines (VM) and containers (C) located in the virtual machines and forming a virtual network. This infrastructure implements the proposed distributed computing placement approach in order to make the right choice in allocating processors to distributed computing processes with the aim of minimizing the Makespan.

4.2. Problem description

4.2.1. Problem statement

In general, placement is used to assign computation processes to the processors of the compute nodes. In this work, it was useful to carry out a placement that minimized the Makespan while taking into account the state of the nodes, the dynamism of the infrastructure and the flow of calculations.

4.2.2. Problem formulation

Consider the following notations:

n: the number of distributed computations;

m: the number of physical nodes which virtual nodes will be assigned tasks in order to minimize the Makespan;

$T = \{ T_1, T_2, \ldots, T_n \}$: a set of n distributed computations;

n_i : the number processes of task T_i, with = 1. . n;

$\delta = \{ \delta_1, \delta_2, \ldots, \delta_n \}$: a set, where δ_i=1 if distributed computation T_i is selected, or 0 otherwise;

P_{ij} : the process j of task T_i;

$I(P_{ij})$: the function that gives the number of instructions of process j:1.. n_i of distributed computation T_i;

$M = \{ M_1, M_2, \ldots, M_m \}$: a set of *m* physical nodes;

m_k: the number of virtual nodes of M_k, k = 1..m;

$V = \{ V_{lr}, l = 1..m, r = 1..m_k \}$: a set of virtual nodes where V_{lr} is a virtual node r of physical node M_l;

C_{lr} : the cores number of virtual node V_{lr}.

F_{lrs} : the frequency of core *s* of virtual node V_{lr}, *s*=1..C_{lr}.

NB: with the use of containers, you can have several virtual nodes on the same physical machine which have different core frequencies from one virtual node to another. We have a very fine-grained granulation of processing units. For Example, we can have a machine that has 5 GHz of frequency where tree virtual nodes (Docker containers) can be tunes with the frequencies 2 GHz, 1 GHz and 1 GHz respectively.

Let θ be the function that determines if a process is selected and affected or not. θ is defined as follow:

$$\theta(i, j, l, r, s) = \begin{cases} 1 \text{ if process j of } T_i \text{ is affected to core s of virtual node r of physical node l} \\ 0 \text{ else} \end{cases}$$

The execution time of process P_{ij} of T_i on core *s* of virtual node V_{lr} is given by the formula: $\frac{I(P_{ij})}{F_{lrs}}$

According to the above notation, the objective function is given by:

$$\text{Minimize : } Z(T) = \begin{cases} \max(\bigcup \delta_i max\,(\bigcup \theta(i,j,l,r,s)\ \frac{I(P_{ij})}{F_{lrs}})) \\ \text{with } i = 1..n \quad j = 1..n_i \quad l = 1..m \quad r = 1..m_k \quad s = 1..C_{lr} \end{cases}$$

Figure 8

Subject to:

1. For all T_i computation:

(a) $n_i > 0$

(b) and for any process P_{ij} of T_i, $I(P_{ij}) > 0$

2. For any virtual node V_{lr} of any physical node l:

(a) $C_{lr} > 0$

(b) and for any core or processor s of V_{lr} , $F_{lrs} > 0$

3. The number of processes affected to cores of each virtual node V_{lr} should not exceed the number of cores of that node:

$$\sum_{i=1}^{n} \delta_i \sum_{j=1}^{n_i} \sum_{s=1}^{C_{lr}} \theta(i,j,l,r,s) \leq C_{lr} \ \ where\ l\ and\ r\ are\ fixed$$

4. A computation T_i is affected if all its processes are affected:

$$\sum_{j=1}^{n_i} \sum_{k=1}^{m} \sum_{l=1}^{m_k} \sum_{r=1}^{C_{lr}} \theta(i,j,l,r,s) = n_i$$

4.3. Methodology

To do the placement of the distributed computations, The processes in general is as follows:

1. Find and determine a solution, which is in fact a group of distributed computations that can be assign and executed
2. If the solution is empty, go to step 4, otherwise improve the solution to minimize Makespan.
3. Run the solution found
4. Check whether there are still any unassigned computations; if there are any unassigned computations, go to step 1; otherwise, exit as all computations are assigned and executed.Those different steps are summarized in the diagram given in Figure2

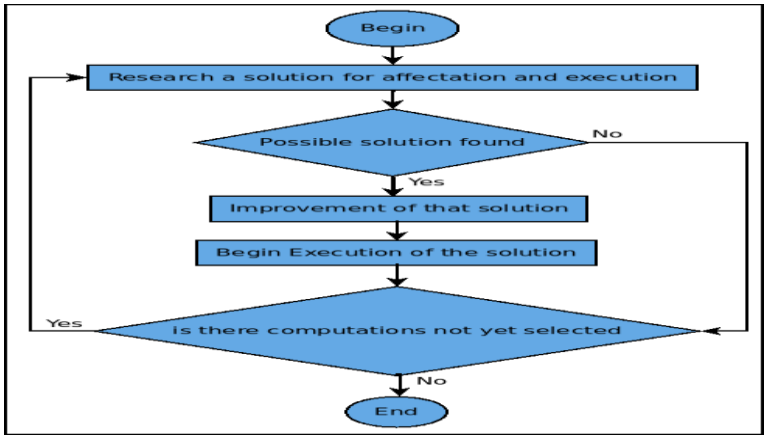

Figure 9: Overview of the proposed placement approach

More specifically, the following section presents each part of the placement strategy proposed below.

4.3.1. Research for a solution for assignment and execution

In the search for a solution which is in fact a group of distributed calculations which can be assigned and executed taking into account the objective and the associated constraints, it has been judicious to divide this part into several sub-parts which are:

- Classification of distributed computations and virtual nodes;
- Assignment of distributed computations to computation nodes.

Classification of distributed computations and virtual nodes

The classification of distributed computations and virtual nodes enables nodes and computations to be organized in order to facilitate the choice of distributed computations and computation nodes during the distributed computation placement phase. To do this, follow these steps:

- Calculate the order order(T_i, *I*) and order(T_i, *n*) which are the orders of the computation T_i respectively according to the average of the number of instructions of all processes of each computation of List_Calc and also of their number of processes
- calculate the order order(V_{lr}, *F*) and order(V_{lr} ,C) are the orders of the virtual node V_{lr} respectively as a function of the average frequency of all the frequencies of each virtual node and also their number of cores
- order distributed computations and virtual nodes according to these orders

Here's the algorithms (Algorithm 1 and Algorithm 2) that explain it:

Algorithm 1 Classification of computations

for *each* $T_i \in List\_Calc$ **do**
  $order(T_i, I) = 0; order(T_i, n) = 0$
  $Average(T_i, I) = \frac{1}{n_i} \sum_{k=1}^{n_i} I(P_{ik})$
  for *each* $T_j \in List\_Calc$ **do**
    $Average(T_j, I) = \frac{1}{n_j} \sum_{k=1}^{n_j} I(P_{jk})$
    if $Average(T_i, I) > Average(T_j, I)$ **then**
      $order(T_i, I) + +$
    end if
    if $n_i > n_j$ **then**
      $order(T_i, n) + +$
    end if
  end for
  $order\_glo(T_i) = order(T_i, I) + order(T_i, n)/2$
end for
Sort tasks in List_Calc *by their* order_glo *in descending order*

Figure 10

Algorithm 2 Classification of virtual nodes

for *each* $V_{lr} \in List\_VN$ **do**
  $order(V_{lr}, F) = 0; order(V_{lr}, C) = 0$
  $Average(V_{lr}, F) = \frac{1}{C_{lr}} \sum_{r=1}^{C_{lr}} F_{lrs}$
  for *each* $V_{du} \in List\_VN$ **do**
    $Average(V_{du}, F) = \frac{1}{C_{su}} \sum_{r=1}^{C_{du}} F_{dus}$
    if $Average(V_{lr}, C) > Average(V_{du}, C)$ **then**
      $order(V_{lr}, F)++$
    end if
    if $C_{lr} > C_{du}$ **then**
      $order(V_{lr}, C)++$
    end if
  end for
  $order\_glo(V_{lr}) = order(V_{lr}, F) + order(V_{lr}, C)/2$
end for
Sort nodes in List_VN *by their* order_glo *in descending order*

Figure 11

Assignment of distributed computations to compute nodes

To assign the classified distributed computations to the ordered nodes, assign each process of each distributed computation to a core of the virtual nodes while respecting the various constraints given. Here is an algorithm that does just that:

Algorithm 3 Assignment of computations to virtual Nodes

$proc\_coeur\_aff = \emptyset$
$List\_Calc\_aff = \emptyset$
for *each* $T_i \in List\_Calc$ *And constraint* (1) *is verified* **do**
  for *each* $V_{lr} \in List\_VN$ *And constraint* (2) *is verified* **do**
    for j *in range*(n_i) *And* s *in range*(C_{lr}) **do**
      if *constraint* (3) *is verified* **then**
        $Assign(P_{ij}, s)$
        $proc\_coeur\_aff = proc\_coeur\_aff \bigcup (P_{ij}, s)$
        $Time(P_{ij}, s) = \frac{I(P_{ij})}{F_{lrs}}$
      end if
    end for
  end for
end for
if *constraint* (4) *is not verified* **then**
  $Cancel\ Assignment\ of\ all\ process\ of\ computation\ T_i$
else
  $Time(T_i) = max(\bigcup Time(P_{ij}, s))$
  $List\_Calc\_aff = List\_Calc\_aff \bigcup T_i$
end if
$Time\_glo\_execution = max(\bigcup Time(T_i))$

Figure 12

4.3.2. Improving the solution obtained and launching the execution of the assigned computations

As part of Improving the solution obtained and launching the execution of the assigned computations, it has the algorithm 4 and algorithm 5 below:

Algorithm 4 Improvement of global execution time of Solution

```
for each (P1_ij, s1) ∈ proc_coeur_aff do
    for each (P2_ij, s2) ∈ proc_coeur_aff do
        if Time_glo_execution is reduced then
            permut (P1_ij, P2_ij) to the resources (s1, s2)
        end if
    end for
end for
```

Figure 13

Algorithm 5 Execution of Assigned computations

```
for all task T ∈ List_Calc_aff do
    Run(T)
    if a task T_i is completed and there are computations not yet affected then
        Research new solution of computations to assign to virtuals nodes
        Assign solution to virtual nodes
        Improve global execution time of solution
        Add solution to List_Calc_aff
    end if
end for
```

Figure 14

The improved solution obtained consists of swapping the distributed computation processes assigned to the cores of the virtual nodes in order to reduce the Makespan while respecting the associated constraints. The algorithm that does this is algorithm 4. After the first enhancement, the following steps are carried out to enable all the distributed computations to be executed:

- Start execution of affected computations
- assign new computations after an affected computation has finished executing
- improve the Makespan of these new computations
- add these affected computations to the list of affected computations

The algorithm that clarifies this is algorithm 5. In this algorithm:

- Order(T_i, I) and order(T_i, N) are the orders of the computation Ti respectiveaccording to to the average of the number of instructions of all processes of each computation of List_Calc and also their number of processes;
- Order(V_{lr}, F) and order(V_{lr}, C) are the orders of the virtual node V_{lr} respectively as a function of the average of the frequency of all the frequencies of each virtual node and also of their number of cores

4.4. Experimentations

During the experimentation phase, the simulation are done by using different approaches on a varied set of nodes and virtual infrastructures and performed distributed computations with different characteristics. The approaches used are:

- algorithm Min-Min[14,15]
- algorithm Min-max[15,16]
- algorithm Improved Min-min[15]
- algorithm Optimized Max-min [17]
- algorithm Optimized Min-Min [18]
- Proposed approach

These different approaches are tested in the following 07 scenarios:

- **scenario 1**: small infrastructure and a distributed computation has exactly one process and a node has exactly one processor or core: With this scenario, the infrastructure is small, each distributed computation is executed by a single node and each node can only execute a single distributed computation;
- ***scenario 2:*** small infrastructure and a distributed computation on several nodes: With this scenario, the infrastructure is small and each distributed computation requires several nodes for its execution;
- ***scenario 3:*** small infrastructure and one node for several distributed computations: With this scenario, the infrastructure is small in terms of the number of compute nodes and each node has the capacity to execute several distributed computations;
- **scenario 4:** large infrastructure and a distributed computation has exactly one process and a node has exactly one processor or core: With this scenario, the infrastructure is large, each distributed computation is executed by a single node and each node can only execute a single distributed computation;
- **scenario 5:** large infrastructure and a distributed computation on several nodes: With this scenario the infrastructure is large and each distributed computation requires *several nodes for its execution.*
- **scenario 6:** large infrastructure and one node for several distributed computations: With this scenario, the infrastructure is large in terms of the number of computation nodes and each node has the capacity to run several distributed computations.
- **scenario 7:** large infrastructure, one node for several distributed computations and one distributed computation on several nodes

Here is a table showing the characteristics of the distributed computations and virtual nodes according to these scenarios:

Table 1: Characteristeristics of nodes and comutation distributed according to the scenarios

	Characteristics of virtual nodes			Characteristics of distributed computations		
Scenario types	Number of virtual nodes	Number of cores per node	Frequency (Mhz) per node	Number of distributed computations	Process number per computation	Number of elementaries instructions (Millions) per process
1	50	1	50-3000	5-500	1	100000-10000000
2	50	1-5	50-3000	5-500	20-35	100000-10000000
3	50	20-35	50-3000	5-500	1-5	100000-10000000
4	500	1	50-3000	5-500	1	100000-10000000
5	500	1-5	50-3000	5-500	20-35	100000-10000000
6	500	20-35	50-3000	5-500	1-5	100000-10000000
7	500	1-35	50-3000	5-500	1-35	100000-10000000

4.5. Presentation and analysis of results obtained

Presentation of the results obtained for the different scenarios

The figures below show the Average Makespan obtained for 10 instances of scenarios 1, 2, 3, 4 , 5,6 et 7 described above.

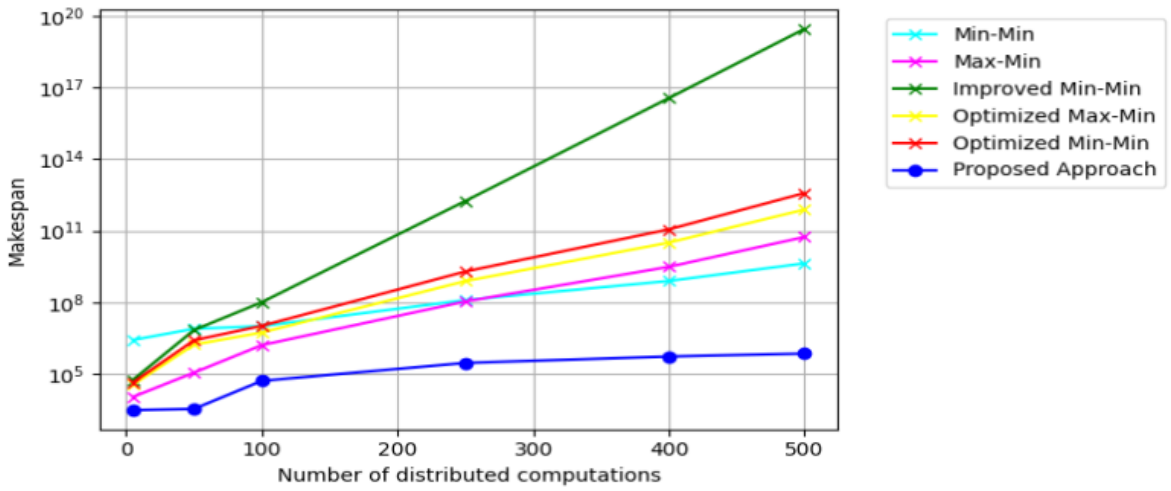

Figure 1: Results of Average Makespan for 10 instances of scenario 1

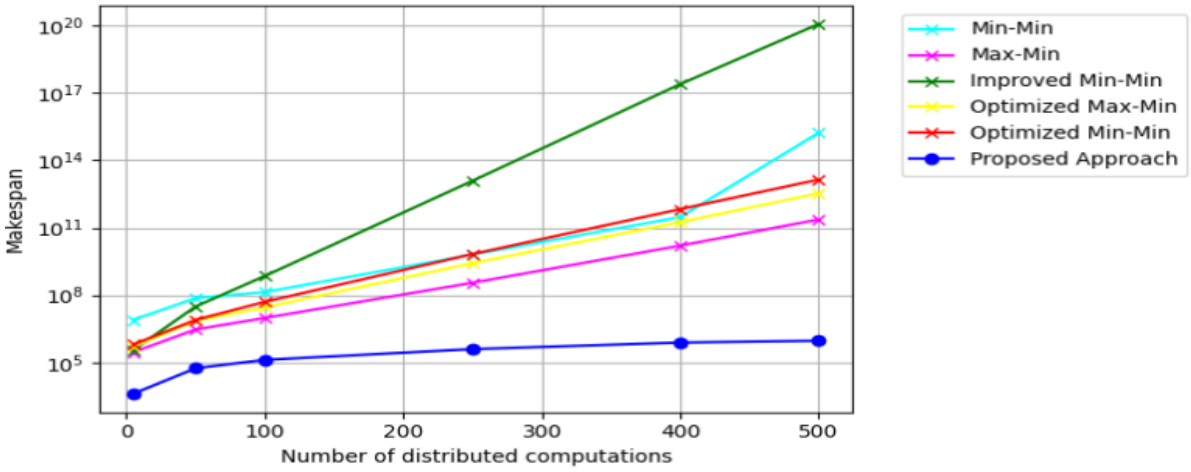

Figure 2: Results of Average Makespan for 10 instances of scenario 2

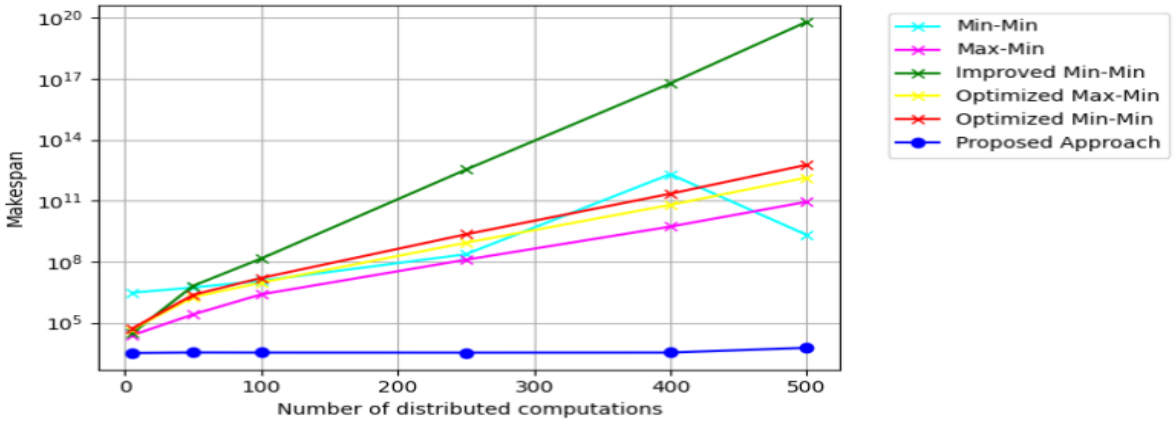

Figure 3: Results of Average Makespan for 10 instances of scenario 3

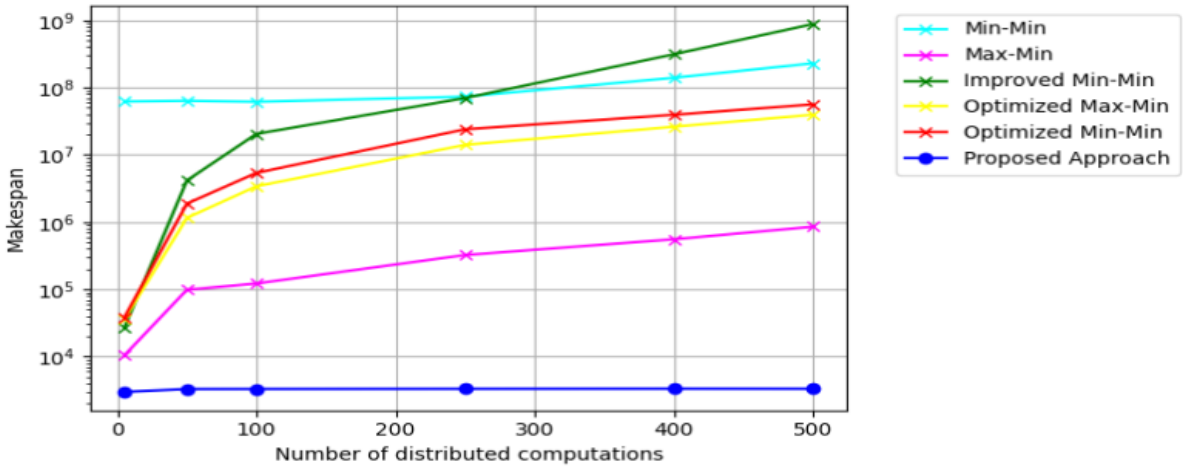

Figure 4: Results of Average Makespan for 10 instances of scenario 4

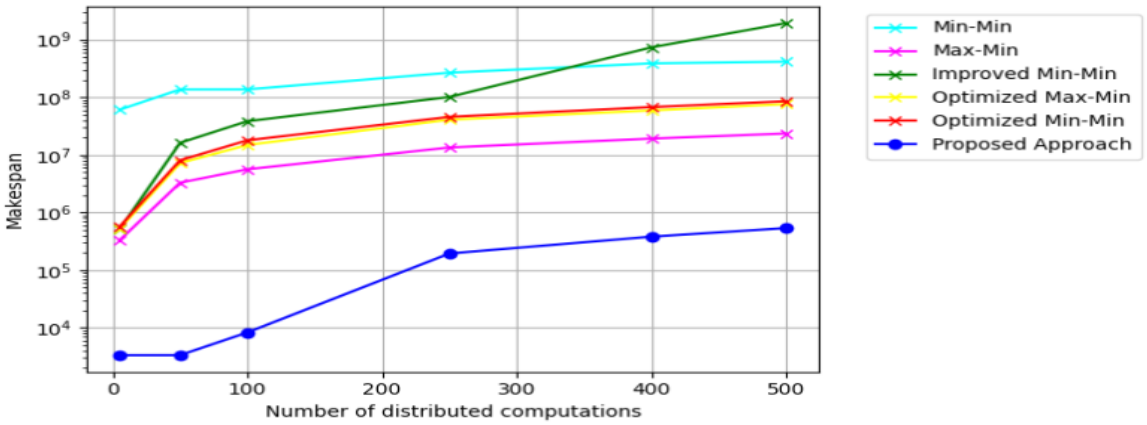

Figure 5: Results of Average Makespan for 10 instances of scenario 5

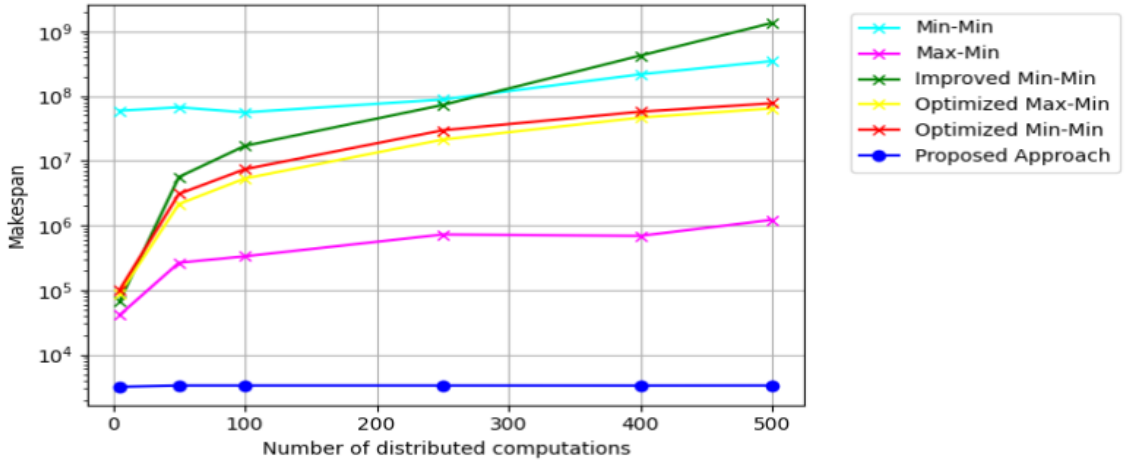

Figure 6: Results of Average Makespan for 10 instances of scenario 5

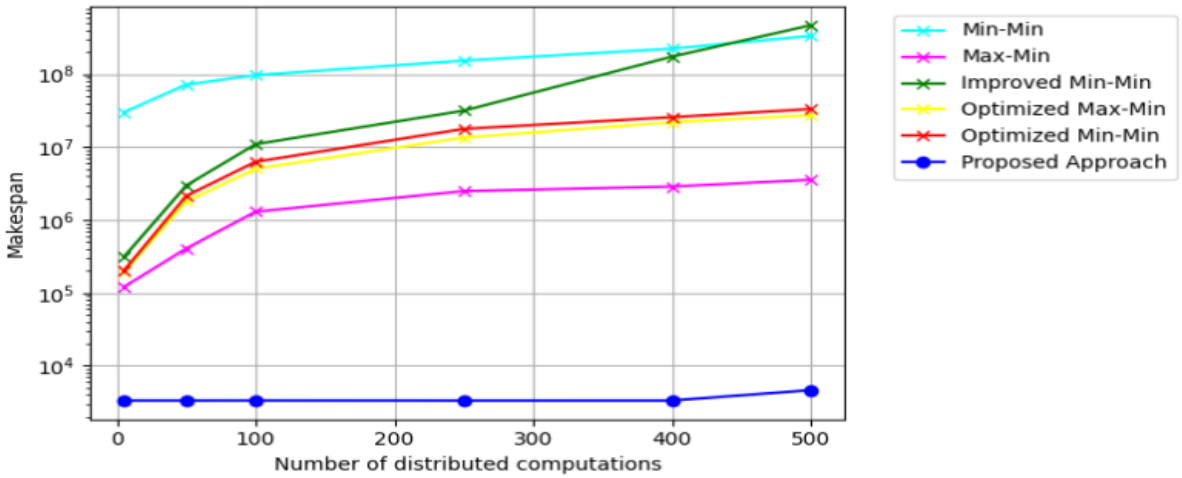

Figure 7: Results of Average Makespan for 10 instances of scenario 7

Analysis of the results obtained for the different scenarios

An analysis of the Makespan results for the different scenarios reveals the following:

1. In scenarios 1, 2 and 3, the Makespan increases significantly as a function of the number of distributed computations (5, 50, 100, 250, 400, 500) for the other approaches (this increase is greater for the Improved Min-Min algorithm) but for our approach this Makespan increases slowly and comparably in scenario 1 and almost zero in scenario 2.
2. In scenarios 4, 5, 6 and 7, the different Makespan values are comparable and sometimes almost stable for each approach, whatever the number of computations distributed. This quasi-stability in execution time is more visible for our approach, especially in scenarios 4, 5 and 7.
3. Although each distributed computation has a single process and each node has a single core in scenarios 1 and 4, our approach still achieves better makespan than the other approaches.
4. In all the different scenarios, it also emerges that our approach brings a considerable and unequal gain in Makespan whatever the number and characteristics of the distributed computations and whatever the number and characteristics of the computation nodes.
5. The effectiveness of our approach stems from its ability to make better investments that minimize Makespan and optimize the use of resources compared with other approaches.

4.6. Discussions

Approaches presented in the literature such as Min-Min, Max-Min, Improved Min-Min, Optimized Max-Min and Optimized Min-Min have shown their limitations in minimizing.

In scenario 1, the approach proposed in this paper has the lowest Makespan of at most 10^5 compared to the other approaches throughout the increase in the number of distributed computations, followed by the Max-Min, Min-Min, optimised Max-Min and optimised Min-Min approaches. The improved Min-Min approach has by far the highest Makespan in this scenario. Furthermore, with the proposed approach, the Makespan evolves very slowly with the increase in distributed computation compared to the other approaches in this scenario. This makes this approach an efficient and fine-tuned resource-use approach for placing distributed computations in an environment with fewer resources and more workloads.

In scenario 2, we reach almost the same conclusion as in scenario 1. It emerges that, compared with other approaches, the proposed approach manages to achieve economical, fine-grained and efficient placement despite the fact that the number of processes per distributed calculation may be well above the number of cores per calculation node and that the calculation environment does not have enough resources. This leads to the conclusion that this approach is economical in managing resource consumption when placed in a distributed system with few resources.

In scenario 3, the approach proposed in this paper has the lowest and quasi-stable Makespan of less than 10^3 compared to the other approaches throughout the increase in the number of distributed computations. The other approaches apart from improved Min-min have Makespans varying from 10^3 to 10^{13} as the number of

distributed computations increases. The improved Min-min approach has by far the highest Makespan in this scenario. Thus, the proposed approach is able to exploit the sufficiently large number of cores available per compute node compared to the number of processes per distributed compute to perform targeted and efficient placement on a distributed system with few compute nodes.

In scenario 4, the approach proposed in this paper has the lowest Makespan and is quasi-stable around 10^2 compared with the other approaches throughout the increase in the number of distributed calculations. This is followed by the Max-min approach whose Makespan varies from 10^4 to 10^6. The other approaches have Makespans in the range $[10^5, 10^9]$. Thus, the proposed approach is able to exploit the sufficiently large number of cores per compute node compared to the number of processes per distributed computation to perform targeted and efficient placement on a distributed system with few compute nodes. Therefore, in a compute environment of manageable size and with a number of distributed compute processes equivalent to the number of cores per node, the proposed approach makes more efficient use of resources to perform placement compared to existing approaches.

In scenario 5, the approach proposed in this paper has the Makespan in the interval $[10^2, 10^6[$ while for the other approaches it is in the interval $[10^5, 10^9]$ throughout the evolution of the number of distributed computations. This shows that this approach is much more efficient in the use of resources during placement. Therefore, in a large-scale cluster environment and with the number of processes per distributed computation much larger than the number of cores per node, the proposed approach makes more efficient use of resources for placement than existing approaches when there are more resources.

In scenario 6, the approach proposed in this paper has the Makespan in the interval $[10^2, 10^6[$ while for the other approaches it is in the interval $[10^5, 10^{10}[$ throughout the increase in the number of distributed computations. This shows that this approach is largely more efficient in the use of resources during placement. As a result, in a distributed environment of manageable size and with the number of processes per distributed computation much larger than the number of cores per node, the proposed approach makes more efficient use of resources to perform placement compared to existing approaches.

In scenario 7, the approach proposed in this paper has the lowest and almost stable Makespan around 10^2 compared to the other approaches throughout the increase of the number of distributed computations while the Makespan of the other approaches are in the range $[10^5, 10^9[$. This shows that this approach is much more efficient in the use of resources during placement. As a result, in a cluster environment of manageable size and with the number of processes per distributed computation as large as the number of cores per node, the proposed approach makes more efficient use of resources for placement than existing approaches.

It appears that the proposed approach presented in this paper adapts better to variations in the characteristics and number of distributed computations and also to variations in the characteristics and number of computation nodes in the placement of distributed computations. Our approach optimizes better the placement of each distributed computation by allocating appropriate computation units to its processes in order to minimize the Makespan. This makes our approach a serious and interesting one for the placement of distributed computations

on compute nodes.

By comparing the approach proposed in this work with several other existing approaches to the placement of distributed computations, it emerges that this proposed approach has the particularity of placing distributed computations in several situations while minimizing the Makespan. Therefore, with this approach, our infrastructure can perform a placement of distributed computations while optimizing the use of resources and, above all, minimizing the Makespan.

5. Conclusion and Perspectives

The study in which we are involved aims to place distributed computations in a virtual environment. The main interest is to know how to place distributed computations in order to minimize the Makespan. To achieve this objective, we first carried out a study of the state of the art in distributed computation and systems, virtualization, placement, existing approaches and finally proposed a virtual infrastructure that integrates placement mechanisms that minimize the Makespan of distributed computation and improve resource utilization. This minimization of Makespan is done by using an approach that explores the loads of the tasks, the loads of their processes, the performance of the nodes and the performance of their different cores in order to place each process of each task on a core of a compute node that takes into account the performance of each node core and the loads of each process of each task. The experiments carried out gave very encouraging results. And after every simulation against a distributed system and distributed calculations, we can directly use the structure obtained to make a placement on this system in real time. Extending this study to connected objects is also a major challenge.

References

[1] Nadiminti, K.; De, A.M.D.; Buyya, R. *Distributed systems and recent innovations:* Challenges and benefits. InfoNet Magazine 2006, 16, 1–5.

[2] Yenke, B.O.; Abba Ari, A.A.; Dibamou Mbeuyo, C.; Voundi, D.A. “Virtual Machine Performance upon Intensive Computations”. *GSTF Journal on Computing (JoC)* 2015, 4. https://doi.org/10.5176/2251-3043_4.3.336.

[3] Stuart, T.A. *Introduction to Distributed Systems*, 1st ed.; Prentice Hall, 1994.

[4] Chanaka, F. “ The Evolution of Distributed Systems”. https://dzone.com/articles/the-evolution-of-distributed-systems, 2018, [Accessed 12 October 2019].

[5] Binason, A. “ IDJ : Une Breve Histoire de l’Ordinateur”. https://babilown.com/2010/02/01/idj-une-brve-histoire-de-lordinateur/, February 2010, [Accessed 15 November 2019].

[6] Rodrigue, C.N. “Environnement d’exécution pour des services de calcul à la demande sur des grappes mutualisées”. Theses, Université de Grenoble, June 2012.

[7] Hyungro, L. “Virtualization basics: Understanding techniques and fundamentals”. School of Informatics and Computing Indiana University 2014, 815.

[8] Phelep, J. “La Conteneurisation”. https://phelepjeremy.wordpress.com/2017/06/21/la-conteneurisation/ , June 2017,[Accessed 10 January 2021].

[9] Meriem, M. " Placement Dynamique de Tâches dans une Grille de Calcul". Theses, Université d'Oran, June 2012.

[10] Braun, T.D.; Siegel, H.J.; Beck, N.; Bölöni, L.L.; Maheswaran, M.; Reuther, A.I.; Robertson, J.P.; Theys, M.D.; Yao, B.; Hensgen, D.; et al. " A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems". *Journal of Parallel and Distributed Computing* 2001, 61, 810–837. https://doi.org/https://doi.org/10.1006/jpdc.2000.

[11] Boyer, W.F.; Hura, G.S. "Non-evolutionary algorithm for scheduling dependent tasks in distributed heterogeneous computing environments". *Journal of Parallel and Distributed Computing* 2005, 65, 1035–1046. https://doi.org/https://doi.org/10.1016/j.jpdc.2005.04.017

[12] Sinnen, O. *Task scheduling for parallel system; Wiley Series on Parallel and Distributed Computing*, Wiley-Interscience, 2007.

[13] Gamboa dos Santos, C. "Problématique du placement de tâches dans MeDLey". Research Report RR-3256, INRIA, 1997

[14] Ali, S.; Siegel, H.; Maheswaran, M.; Hensgen, D.; Ali, S. "Modeling task execution time behavior in heterogeneous computing systems", *Tamkang J. Science and Engineering* 2000, 3, 195–207.

[15] Singh, L.; Agarwal, N." An Improved Min-Min Task Scheduling Algorithm with Grid Utilization and Minimized Makespan". *International journal of computers amp;amp; Technology* 2015,14, 5960–5966. https://doi.org/10.24297/ijct.v14i8.1860

[16] Maheswaran, M.; Ali, S.; Siegel, H.J.; Hensgen, D.; Freund, R.F. "Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems". *Journal of Parallel and Distributed Computing* 1999, 59, 107–131. https://doi.org/https://doi.org/10.1006/jpdc.1999.1581.

[17] Konjaang, J.; Fahrul, H.; Muhammed, A. "An optimized max-min scheduling algorithm in cloud computing". *Journal of Theoretical and Applied Information Technology* 2017, 95, 1916–1926.

[18] Sallar, Salam, M.; Badel, R.; Nashat, Salih abdulkarim, A.; Rafi Faraj, A.; Reham, A.A.; Abdullah, M.; Mohd, D. "optimized MIN-MIN Task scheduling algorithm for scientific workflows in a cloud environment". *Journal of Theoretical and Applied Information Technology* 2023, 100, 480–506.