

jMetal and *MFHS* collaboration for task scheduling optimization in heterogeneous distributed system

Abdelhamid Khiat   1

¹Networks and Distributed Systems Division, Research Center on Scientific and Technical Information, 05, Rue des 3 frères aissou - Ben Aknoun - Algiers, Algeria

Received 16 September 2023, Accepted 06 January 2024, Published 20 January 2024


Abstract. Task scheduling in distributed computing architectures has attracted considerable research interest, leading to the development of numerous algorithms aiming to approach optimal solutions. However, most of these algorithms remain confined to simulation environments and are rarely applied in real-world settings. In a previous study, we introduced the *MFHS* framework, which facilitates the transition of scheduling algorithms from simulation to practical deployment. Unfortunately, *MFHS* currently offers a limited selection of scheduling heuristics. In this work, we address this limitation by presenting the *MFHS_jMetal* framework, integrating the extensive task scheduling algorithms available in the well-established *jMetal* framework. Our implementation demonstrates the successful expansion of available scheduling algorithms while preserving the core characteristics of *MFHS* bridging the gap between theoretical models and real-world deployment.

Keywords: Distributed computing, Scheduling, *MFHS*, *jMetal*).

1 Introduction

In general, the task scheduling problem in heterogeneous distributed environments is recognized as an NP-Hard problem [6], meaning that, to date, no solution has been found to efficiently determine the optimal mapping of a set of tasks to a set of resources in polynomial time as the problem size increases. This challenge has spurred significant efforts within the scientific community to develop algorithms that can provide solutions closely approximating the optimal solution within a reasonable timeframe. Consequently, a substantial number of heuristics have been proposed in the existing literature.

Most of the proposed scheduling heuristics are evaluated through simulation, employing specific evaluation tools. Among the available tools for simulation-based evaluation, we mention *jMetal* [5] [13], which we utilized in this study. *jMetal* has been steadily gaining popularity in the field of multi-objective optimization as an open-source framework. This rise in popularity can be attributed to its extensive library of pre-implemented heuristics, including powerful algorithms such as NSGA-II [4], PAES [11], and SPEA2 [19].

 Corresponding author. Email: a.khiat@dtri.cerist.dz

One of the major issues with *jMetal*, as well as many other simulators, is the absence of a mechanism for seamlessly transitioning evaluated algorithms from simulation to real-world deployment. Our 'MFHS' framework, previously introduced in our work [9], addresses this challenge by enabling automatic switching from simulation to practical deployment. However, a significant limitation of MFHS is its insufficient number of implemented algorithms, which can be remedied by proposing a solution that facilitates interaction with *jMetal*, a framework renowned for its extensive collection of task scheduling algorithms.

To overcome the limitations of both the *jMetal* and MFHS frameworks, this paper introduces a solution named MFHS_*jMetal*, which fosters collaboration between these two tools, complementing each other's functionalities. Our proposed integration is built upon an added layer within MFHS, facilitating interaction with *jMetal*, particularly in cases where the required scheduling heuristics are unavailable in MFHS. Consequently, MFHS_*jMetal* empowers a wide array of task scheduling algorithms (previously untested in real-world scenarios) to be automatically deployed in practical environments. This approach enhances our understanding of algorithm behavior when employed in real-world applications.

The remainder of this paper is structured as follows: Section 2 provides an overview of related work, while Section 3 offers a concise introduction to the *jMetal* and MFHS frameworks. Section 4 delves into the specifics of the proposed MFHS_*jMetal* framework. Subsequently, Section 5 presents experimental findings through a case study. Finally, Section 6 concludes the article and outlines potential avenues for future enhancements.

2 Related work

In the literature, many efforts have been made to propose solutions that assist both researchers and developers in simulating and deploying scheduling algorithms, thereby facilitating the study of algorithm behavior and enabling comparisons with existing heuristics.

In the remainder of this section, we present some simulators and frameworks. Simulators allow the deployment and evaluation of scheduling algorithms in simulated distributed computing environments, while frameworks enable the deployment of scheduling algorithms in real-world distributed architectures.

SimGrid [3] is a versatile simulator that provides essential functionality for simulating distributed applications in heterogeneous environments, offering a suitable platform for heuristic evaluation, prototyping, and the development and enhancement of grid applications. It can be employed for various large-scale systems, including Grids, P2P systems, and Cloud environments.

GSSIM [1] and DCWorm citekurowski2013dcworms are two simulators developed to support experimental studies of resource allocation and scheduling policies in distributed systems. DCWorm extends GSSIM by incorporating additional features focused on energy consumption and efficiency studies.

CloudSim [7] is a widely recognized tool for modeling and simulating cloud computing environments. It enables users to define key characteristics of data centers, including the number and specifications of hosts, available storage, network topology, and usage patterns. Many other simulators, such as ElasticSim, CloudAnalyst, and Dynamiccloudsim, have been built upon CloudSim as foundational pillars.

iFogSim [8] is a toolkit designed for modeling and simulating resource management techniques in IoT, Edge, and Fog computing environments. Building upon the event simulation

functionalities of CloudSim, MyiFogSim extends iFogSim to include virtual machine migration simulation.

Núñez et al. have proposed a flexible simulator called iCanCloud [15], is based on SIMCAN [14], and it is a flexible simulator designed for modeling HPC architectures. Both SIMCAN and iCanCloud are open-source simulators developed in C++. They support parallel experiments and the addition of various adapted MPI libraries and POSIX-based APIs for simulating new applications.

OpenStack Neat [2] is an open-source framework built upon the OpenStack platform. It focuses on dynamic VM consolidation scheduling in cloud data centers. This framework functions as a transparent add-on to OpenStack, allowing for use without modification of the original installation or specific configurations.

CLOUDRB [16], proposed by Somasundaram and Govindarajan, serves as a cloud resource broker for scheduling and managing High-Performance Computing (HPC) applications in Science Cloud. It integrates a deadline-based job scheduling policy with a particle swarm optimization-based resource scheduling mechanism, aiming to minimize both cost and execution time to meet user-specified deadlines.

CometCloud [10] is a cloud framework designed for autonomic workflow management, addressing changing computational and Quality of Service (QoS) requirements. It aims to create a virtual computational cloud infrastructure that seamlessly integrates local computational environments and public cloud services on-demand, providing abstractions and mechanisms supporting various programming paradigms and real-world applications.

A general cross-layer Cloud scheduling framework for IoT tasks [18] dynamically selects appropriate algorithms based on specific task criteria. Numerous experiments using the CloudSim simulator have been conducted to evaluate this framework.

Maxinet [17] offers distributed emulation of software-defined networks. Additionally, EmuFog [12] is an emulator for Fog Computing environments built atop MaxiNet. EmuFog supports the emulation of real applications and is designed to accommodate large-scale, scalable typologies. It is an open-source tool that encourages extensibility, allowing developers to override components as needed to adapt to their requirements."

3 Background

As the primary objective of the framework presented in this paper is to facilitate collaboration between the *MFHS* and *jMetal* frameworks, it is essential to provide a more in-depth explanation of the functioning of both these frameworks. This will enhance comprehension of the subsequent sections of the paper.

A. jMetal

jMetal is a Java-based framework developed from scratch, oriented towards addressing multi-objective optimization problems using heuristics. It is designed for use in various disciplines, including task scheduling problems, which are the focus of the present work. *jMetal* is built to be portable, extensible, flexible, and easily reusable.

jMetal has evolved into a Maven project*. In our work, we have concentrated on version 5.0 of *jMetal*, which is organized into four distinct packages readily available on the Maven

*Maven Project, <https://maven.apache.org/>

Central Repository **:

jmetal-core: This package encompasses classes and interfaces related to the core architecture, along with various utilities, including quality indicators.

jmetal-algorithm: Provides implementations of widely recognized meta-heuristics, including NSGA II, NSGA III, SPEA2, ESPEA, and more.

jmetal-problem: Encompasses implementations of well-established problems, both single-objective and multi-objective benchmarks.

jmetal-exec: Contains executable programs that facilitate the configuration and execution of algorithms.

A. MFHS

MFHS offers an appealing feature termed "From Theoretical to Real Deployment." This feature allows for the analysis of scheduling algorithms using theoretical data for simulation purposes, followed by an automatic transition from simulation to actual deployment. Additionally, MFHS is designed for seamless adaptability to various distributed architectures and facilitates easy interaction with external solutions.

Regrettably, as of the present, MFHS has a limited number of scheduling heuristics available. To address this limitation, we have integrated an external solution, jMetal, which already includes a substantial number of task scheduling algorithms.

MFHS functions are realized through six modules:

Resources Discovery: This module discovers available resources along with their characteristics, including estimated upload/download speeds, CPU specifications, and more.

Requests Collector: Responsible for gathering data related to task execution requests, such as data size for upload/download, CPU utilization duration, and the number of required logical CPUs.

Scheduling: This module integrates information collected by the *Resources Discovery* and *Requests Collector* modules. It then utilizes scheduling algorithms to determine resource-task mappings.

Resources Allocation: Performs actual resource allocation based on the assignment matrix obtained from the *Scheduling* module.

Monitoring: This module provides control over the platform during the execution of the task scheduling process. It can identify anomalies, such as loss of connection to a compute node, and archives various monitoring data for future uses.

Behavior Study: Invoked at the conclusion of the scheduling process, this module collects information from all other modules and generates comprehensive reports.

4 Proposed MFHS_jMetal framework

jMetal already contains a significant number of task scheduling algorithms that are primarily suitable for simulation but lack a mechanism for transitioning to real deployment in a distributed environment. Fortunately, MFHS offers an intriguing feature, enabling this transition from theoretical to real deployment. However, MFHS currently has a limited number of scheduling algorithms available.

In this work, we aim to leverage the strengths of MFHS and the jMetal framework while mitigating their respective limitations.

**Maven Central Repository, <http://search.maven.org>

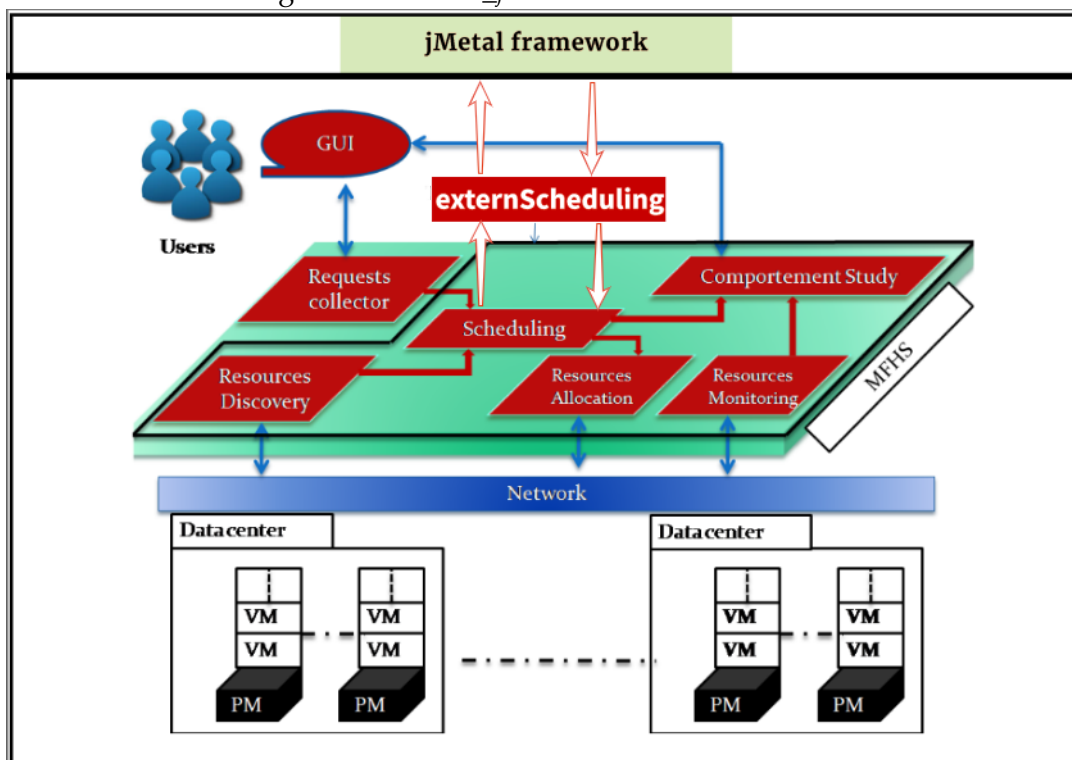
To achieve our objectives, our proposed solution enables the invocation of jMetal from within MFHS, allowing the execution of any task scheduling algorithm that is unavailable in MFHS. This process involves two steps: First, MFHS provides data related to both task and resource characteristics. Second, jMetal returns the mapping of the task set to the resource set based on the selected algorithm. Finally, MFHS continues with the remaining steps, as previously described in Subsection 3.A.

4.1 High level Architecture

Figure 4.1 illustrates the global architecture of our proposed *MFHS_jMetal*. It reveals the addition of an extra layer atop the *Scheduler* module of MFHS. This layer serves as a seamless interface, housing a function named *externScheduling*. This function is triggered when the required scheduling algorithm is unavailable in MFHS. *externScheduling* facilitates interaction with an external framework containing the necessary scheduling algorithms, with jMetal serving as the external framework in this paper. However, it's worth noting that this function can be adapted to call a framework other than jMetal if needed.

jMetal is an open-source project that allows the incorporation of additional functions. In our case, we have introduced a function capable of receiving input data, including information about the task set, resource set, and estimated task execution times on each resource, among other details. After executing a task scheduling algorithm using the provided data, the resulting mapping is made accessible through the added function as output. This output is subsequently collected and utilized by the remaining processes within MFHS.

Figure 4.1: *MFHS_jMetal* Global Architecture



4.2 Call proseeus

Algorithm 1 represents the portion we have added, which is injected after the *Resource Discovery* and *Request Collector* modules, and before the execution of the desired scheduling algorithm. This injected algorithm allows for the execution of the corresponding task scheduling algorithm if it is available within MFHS. If the algorithm is not available, it calls jMetal while sending the necessary data as input. Once the algorithm is executed, the results are collected, corresponding to the possible mappings found. These collected results are saved into two distinct files, namely *Solutions.tsv* and *Objectives.tsv*.

Solutions.tsv contains all dominant solutions, with each line having a corresponding line in *Objectives.tsv*, which contains the values of the objectives to be optimized. Subsequently, MFHS continues its execution based on these two files, which provide sufficient data for the rest of the MFHS process.

Algorithm 1 requires as input the meta-task that includes the set of tasks to be executed and the estimated execution time for each task on each resource. In the algorithm, N and M represent the total number of tasks and the total number of resources, respectively. Additionally, $Root_O_1$ through $Root_O_p$ represent a set of p matrices, each with a size of $n * m$. Here, p denotes the number of matrices needed for calculating the values of the objectives to be optimized. $Root_O_l[i][j]$ represents the estimated value of the l th matrix for task i if executed on resource j .

In the current version of MFHS, following MFHS_jMetal, a maximum of four objectives can be considered. These objectives are: *makespan*, *Resource utilization*, *Cost*, and *Energy consumed*.

Algo represents the name of the algorithm to be executed. Depending on the chosen *Algo*, additional information may be required as input. A detailed description of the required *Otherdata* will be presented in Section 5.

```

Input :  $Root\_O1[n][m], Root\_O2[n][m], \dots, Root\_Op[n][m], Algo, Otherdata$ 
Output: mapping, objective values
if (Algo in MFHS) then
  | Call algo from MFHS;
else
  | if (Algo in jMetal) then
  | | call Algo from jMetal
  | else
  | | exit(0)
  | end
end
Return Affectation

```

Algorithm 1: MFHS_jMetal Global processes

4.3 MFHS_jMetal objectives measurement and representation

Thanks to MFHS and its modularity property, which enables us to easily reuse one or more modules, we have added an additional function atop the *Scheduling* module without making any modifications to other modules.

Assuming we have two algorithms to evaluate before their deployment, Algorithm1MFHS and Algorithm2jMetal. The first one is available within the MFHS framework, while the

second one is available in the jMetal framework. Both jMetal and MFHS require input data, and the nature of the required data may vary depending on the algorithm.

Each algorithm is represented by a corresponding Java class. Consequently, the constructors for Algorithm1MFHS and Algorithm2jMetal are Algorithm2MFHS ($ET[n][m]$, $Cost[n][m]$, $Energy[n][m]$, $Array(input\ variable)$) and Algorithm2jMetal ($ET[n][m]$, $Cost[n][m]$, $Energy[n][m]$, $Array(input\ variable)$). Here, $ET[n][m]$, $Cost[n][m]$, and $Energy[n][m]$ respectively represent the estimated execution time, cost, and energy consumption of each task on each resource. These values change based on the task requirements and resource capacities, independent of the chosen algorithm. *Input variable* encompasses other variables dependent on the algorithm in use, such as selection probability and mutation probability in the case of the GA algorithm.

We assume that $fEnergy$, $fMakespan$, and $fCost$ are the objective functions to be optimized, corresponding to Energy consumed, Total execution time, and Cost, respectively.

$$\begin{aligned} fEnergy &= \sum_{i=0}^{n-1} Energy[R[T_i]] \\ fCost &= \sum_{i=0}^{n-1} Cost[R[T_i]] \\ fMakespan &= \max_{i \in [0..n-1]} (ResTime[T_i]) \end{aligned}$$

We are interested in the non-dominated solutions. The concept of dominated solutions is defined as follows: Each solution s is represented by a tuple ($Energy(s)$, $Cost(s)$, and $Makespan(s)$). Solution s_1 dominates s_2 if $Energy(s_1) \leq Energy(s_2)$, $Cost(s_1) \leq Cost(s_2)$, and $Makespan(s_1) \leq Makespan(s_2)$.

The set of non-dominated solutions is saved in a file called *objectives.tsv*. Each line is written in the following format: $Key_s : Energy(s), Cost(s), Makespan(s)$. Therefore, each line in *objectives.tsv* is associated with a line in another file called *solutions.tsv*, identified by key_s . Each line in *solutions.tsv* is written as a sequence of n values in the following format: $Key_s : R[T_0] R[T_1] \dots R[T_i] \dots R[T_n]$, where $R[T_i]$ represents the resource on which T_i is executed. Key_s is a key associated with each solution.

With the information in *solutions.tsv* and *objectives.tsv*, we can generate various representations to help understand the results and select the most suitable solution for real implementation. The selection of the best solution can be done manually or using a specific program.

Upon completion of the execution of all algorithms and obtaining the *solutions.tsv* and *objectives.tsv* files, the remaining steps of the MFHS process in a real environment can be continued, including resource allocation, monitoring, and behavior studies.

5 Exprementation

To demonstrate the successful integration we have proposed, we began by implementing the layer discussed in Section 4 using JAVA, R, and bash programming languages. Subsequently, we conducted our experimentation using two algorithms: the first one called NSGA II, available exclusively in jMetal, and the second one named InterRC, available in MFHS. It is important to note that the objective of our presented experimentation is not to compare these two algorithms but rather to showcase the effective functioning of the proposed MFHS and jMetal integration.

To initiate the evaluation of InterRC and NSGA II, the file *StartCases.sh* needs to be filled with the two following lines as specified in Table 5.1:

```
./MultiCaseNSGAI.sh 0.05 0.8 250 100 solutions1.tsv objectives1.tsv 512 16
./InterRC.sh 10 500 solutions2.tsv objectives2.tsv 512 16 250 100
```

A section of code implemented in bash was added, named *CallAlgojMetal.sh*, which enables MFHS to interact with jMetal by facilitating the exchange of necessary data.

NSGA II Algorithm	
Parameter	Value
Problem size	512 Task on 16 Resources
Crossover probability	0.80
Mutation probability	0.05
Population size	250
Number of generation size	100
InterRC Algorithm	
Parameter	Value
Problem size	512 Task on 16 Resources
Population size	250
Number of generation size	100

Table 5.1: NSGA II and InterRC input parameters

Figure 5.1 and Figure 5.2 respectively depict extracts from objectives1.tsv and objectives2.tsv. The latter file contains the objective values to be optimized, corresponding to the Pareto front values.

Each line in objectives1.tsv and objectives2.tsv corresponds to a line in solution1.tsv and solution2.tsv, respectively. These solution files contain the assignments that produce the obtained values of the objectives to be optimized.

Figure 5.1: NSGA II Pareto front

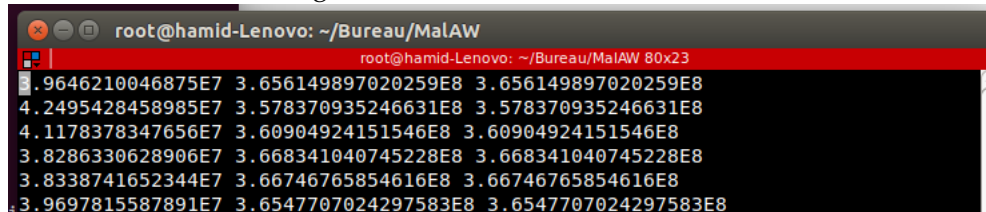
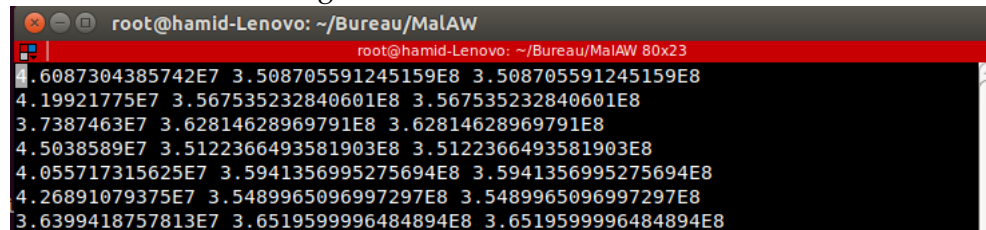


Figure 5.2: InterRC Pareto front



In our experimentation, we evaluated the two algorithms based on three objectives: Energy consumption, Cost, and Total response time. Consequently, MFHS executed CallAlgojMetal.sh with the following inputs: ET[n][m], Cost[n][m], Energy[n][m], and the array representing, in our case, the number of generations, probability of crossover, and probability of mutation.

6 Conclusion and Future workă

In this paper, we have introduced the MFHS_jMetal framework, which involves the integration of a layer on top of the *Scheduling* module of the MFHS framework. This layer enables interaction with the jMetal framework.

Our proposed solution offers the advantages of both MFHS and jMetal while mitigating their respective disadvantages. The integration expands the pool of task scheduling algorithms available to users on one hand and incorporates the "From theoretical to real deployment" feature into jMetal without making extensive changes to MFHS.

Experiments demonstrate the successful implementation of the discussed functions and the achievement of the integration's objectives.

In this work, our experiments focused solely on the scheduling module of MFHS as the primary objective was to validate our proposed idea. Future work may involve the deployment of our solution in a real distributed environment, such as Cloud computing, for more extensive experiments.

Additionally, we have initiated the call from MFHS to jMetal in this paper. Future work may consider the reverse operation, calling MFHS from jMetal. This would be particularly valuable for researchers interested in working with real data collected from real environments.

Declarations

Funding

There are no funding sources to declare for this study.

Authors' contributions

All aspects of this research, including the design, experiments, and manuscript preparation, were solely conducted and completed by Abdelhamid KHIAT.

Conflict of interest

The author declares no conflicts of interest related to this research.

References

- [1] S. BAK, M. KRYSZEK, K. KUROWSKI, A. OLEKSIK, W. PIATEK, AND J. WAGLARZ, *Gssim—a tool for distributed computing experiments*, *Scientific Programming*, **19**(4) (2011), 231–251. [DOI](#)
- [2] A. BELOGLAZOV AND R. BUYYA, *OpenStack Neat: a framework for dynamic and energy-efficient consolidation of virtual machines in OpenStack clouds*, *Concurrency and Computation: Practice and Experience*, **27**(5) (2015), 1310–1333. [DOI](#)
- [3] H. CASANOVA, A. LEGRAND, AND M. QUINSON, *Simgrid: A generic framework for large-scale distributed experiments*, In *Tenth International Conference on Computer Modeling and Simulation*, IEEE, 2008 (uksim 2008), pages 126–131. [DOI](#)

- [4] K. DEB, A. PRATAP, S. AGARWAL, AND T. MEYARIVAN, *A Fast Elitist Multi-Objective Genetic Algorithm: NSGA-II*, IEEE Transactions on Evolutionary Computation, **6** (2000), 182–197. [URL](#)
- [5] J. J. DURILLO AND A. J. NEBRO, *jMetal: A Java framework for multi-objective optimization*, Advances in Engineering Software, **42**(10) (2011), 760–771. [DOI](#)
- [6] M. R. GARY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-completeness*, 1979. [DOI](#)
- [7] T. GOYAL, A. SINGH, AND A. AGRAWAL, *Cloudsim: simulator for cloud computing infrastructure and modeling*, Procedia Engineering, **38**, (2012), 3566–3572. [DOI](#)
- [8] H. GUPTA, A. VAHID DASTJERDI, S. K. GHOSH, AND R. BUYYA, *iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments*, Software: Practice and Experience, **47**(9) (2017), 1275–1296. [DOI](#)
- [9] A. KHIAT, A. TARI, AND T. GUÉROUT, *MFHS: A modular scheduling framework for heterogeneous system*, Software: Practice and Experience, **50**(8) (2020), 1463–1497. [DOI](#)
- [10] H. KIM AND M. PARASHAR, *CometCloud: An autonomic cloud engine*, Cloud Computing: Principles and Paradigms, (2011), pp. 275–297. [DOI](#)
- [11] J. D. KNOWLES AND D. W. CORNE, *Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy*, Evolutionary Computation, **8**(2) (2000), 149–172. [DOI](#)
- [12] R. MAYER, L. GRASER, H. GUPTA, E. SAUREZ, AND U. RAMACHANDRAN, *Emufog: Extensible and scalable emulation of large-scale fog computing infrastructures*, In 2017 IEEE Fog World Congress (FWC), IEEE, 2017, pp. 1–6. [DOI](#)
- [13] A. J. NEBRO, J. J. DURILLO, AND M. VERGNE, *Redesigning the jMetal multi-objective optimization framework*, In Proceedings of the companion publication of the 2015 annual conference on genetic and evolutionary computation, 2015, pp. 1093–1100, . [DOI](#)
- [14] A. NÚÑEZ, J. FERNÁNDEZ, R. FILGUEIRA, F. GARCÍA, AND J. CARRETERO, *SIMCAN: A flexible, scalable and expandable simulation platform for modelling and simulating distributed architectures and applications*, Simulation Modelling Practice and Theory, **20**(1) (2012), 12–32. [DOI](#)
- [15] A. NÚÑEZ, J. L. VÁZQUEZ-POLETTI, A. C. CAMINERO, G. G. CASTAÑÉ, J. CARRETERO, AND I. M. LLORENTE, *iCanCloud: A flexible and scalable cloud infrastructure simulator*, Journal of Grid Computing, **10**(1) (2012), 185–209. [DOI](#)
- [16] T. S. SOMASUNDARAM AND K. GOVINDARAJAN, *CLOUDRB: A framework for scheduling and managing High-Performance Computing (HPC) applications in science cloud*, Future Generation Computer Systems, **34** (2014), 47–65. [DOI](#)
- [17] P. WETTE, M. DRÄXLER, A. SCHWABE, F. WALLASCHEK, M. H. ZAHRAEE, AND H. KARL, *Maxinet: Distributed emulation of software-defined networks*, In 2014 IFIP Networking Conference, pages 1–9. IEEE, . [DOI](#)
- [18] G. WU, W. BAO, X. ZHU, AND X. ZHANG, *A general cross-layer cloud scheduling framework for multiple iot computer tasks*, Sensors, **18**(6):1671, (2018). [DOI](#)

- [19] E. ZITZLER, M. LAUMANN, AND L. THIELE, *SPEA2: Improving the strength Pareto evolutionary algorithm*, TIK-report, 103, 2001. [DOI](#)