# Evolutionary Multi-Objective Task Scheduling for Heterogeneous Distributed Simulation Platform

Xutian He[1][a], Yanlong Zhai[2](✉)[b], Ousman Manjang[2][c] and Yan Zheng[2][d]
[1]*School of Computer Science and Technology, Beijing Institute of Technology, Beijing, China*
[2]*School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing, China*

Keywords: Distributed Simulation, Script Engine, Simulation Task Scheduling, Multi-Objective Optimization, Genetic Algorithm.

Abstract: Most existing distributed simulation platforms lack native support for Python scripts, thereby hindering the seamless integration of AI models developed in Python. Some simulation platforms support script languages like Lua or javascript, but scheduling tasks in heterogeneous simulation platforms that are composed of simulation engine and script engine is a challenging problem. Moreover, conventional task scheduling methods often overlook the simulation time constraints, which are essential for simulation synchronization. In this paper, we propose a Heterogeneous Distributed Simulation Platform (HDSP) that could integrate different script languages, especially Python, to empower the simulation by leveraging intelligent AI models. A Dynamic Multi-Objective Optimization (D-MO) Scheduler is also designed to efficiently schedule simulation tasks that run across heterogeneous simulation engines and satisfy simulation synchronization constraints. HDSP integrates various script engines, enhancing its adaptability to model dynamic simulation logic using different script languages. D-MO Scheduler optimizes Simulation Acceleration Ratio (SAR), Average Weighted Waiting Time (AWWT), and Resource Utilization (RU). The D-MO scheduling problem is characterized as an NP-hard problem, tackled using the NSGA-III algorithm. The simulation time synchronization constraints are implemented through Lower Bound on Time Stamp (LBTS) and lookahead approach. The comparative results and statistical analysis demonstrate the superior efficacy and distribution performance of proposed D-MO Scheduler. The proposed HDSP and D-MO Scheduler significantly boost the capability to support Python-based AI algorithms, and navigate complex scheduling demands efficiently.

## 1 INTRODUCTION

In the realm of computational simulation, the integration of artificial intelligence (AI) algorithms represents a paradigm shift towards more intelligent and adaptable systems(Jawaddi and Ismail, 2024). However, simulation platforms have primarily been developed using traditional programming languages such a C++(Adday et al., 2024; Ierusalimschy et al., 2011) and script languages such as Lua(Chang et al., 2019). Consequently, they often lack direct support for Python, despite its widespread adoption in the AI research community. Therefore, the integration of Python script engine with simulation platforms enables the application of cutting-edge AI algorithms, improving advanced simulation capabilities.

Research efforts to integrate the Python engine into complex simulation platforms have revealed considerable challenges. For instance, Liu et al. managed to extend Python support to Java-based systems, but at the expense of excluding traditional script languages like Lua, known for their established ecosystems(Liu et al., 2021). Wornow et al. redesign an existing medical simulation engine utilizing Python. However, this approach introduced additional workload and restricted its application scope solely to the medical field(Wornow et al., 2023). These limitations highlights the necessity for a more robust approach that embraces Python alongside other script languages. Additionally, the integration of script engines introduces heterogeneity to the simulation platform, requiring consideration of synchronization time constraints. These factors are overlooked by many prevailing task scheduling methods, resulting in in-

[a] https://orcid.org/0009-0005-7334-0799
[b] https://orcid.org/0000-0002-0168-8308
[c] https://orcid.org/0009-0005-9337-2405
[d] https://orcid.org/0009-0008-3083-3931



He, X., Zhai, Y., Manjang, O. and Zheng, Y.
Evolutionary Multi-Objective Task Scheduling for Heterogeneous Distributed Simulation Platform.
DOI: 10.5220/0012814300003758

In *Proceedings of the 14th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2024)*, pages 150-157
ISBN: 978-989-758-708-5; ISSN: 2184-2841

accuracies and inconsistencies in the simulation outcomes. Therefore, there is a pressing demand for task scheduling methods that accommodate simulation time synchronization constraints, and optimize conflicting goals such as makespan and resource consumption(Lu et al., 2020), which is a NP-hard multi-objective optimization (MO) problem.

In this paper, we propose Heterogeneous Distributed Simulation Platform (HDSP) and the Dynamic Multi-Objective Optimization (D-MO) Scheduler, designed to support Python-based AI algorithms while ensuring simulation time synchronization. HDSP addresses the absence of native Python support by integrating multiple script engines, enhancing its adaptability to diverse script languages. The D-MO Scheduler is designed to solve the NP-hard task scheduling problem in HDSP. It optimizes multiple objectives, including Simulation Acceleration Ratio (SAR), Average Weighted Waiting Time (AWWT), and Resource Utilization (RU), all the while accounting for synchronization constraints. The D-MO Scheduler employs NSGA-III(Deb and Jain, 2014) for the scheduling algorithm, and manages the synchronization constraint through Lower Bound on Time Stamp (LBTS) and lookahead approach. This innovation enables the integration of Python-based AI algorithms into simulation agents, and promotes the simulation field towards a more adaptable and intelligent future.

## 2 RELATED WORK

Considering integration of Python engine for simulation platforms, two types of methods are adapted in existing research. The first type involves the construction of simulation platforms entirely in Python, thereby inherently supporting Python scripts. Souza et al. proposed a simulation framework developed in Python to simulate resource management policies in Edge Computing environments(Souza et al., 2023). Chambon et al. developed a Python simulator to model user consumption behavior for water distribution networks(Chambon et al., 2024). However, this approach constrains its application to limited scenarios. The second type extends Python support to existing platforms. D'Aquin et al. developed Python interface to PartMC, a simulation model implemented in Fortran(D'Aquino et al., 2024). Wong et al. proposed a dedicated Python library built to support simulation(Wong et al., 2023). These methods lack versatility because they exclude traditional script languages like Lua, which possess established ecosystems.

The embedding of multiple script engines into simulation platforms introduces heterogeneity, making simulation time synchronization of task scheduling become a great challenge. Heterogeneous Earliest Finish Time (HEFT) is an effective metric for scheduling in heterogeneous systems. Vasilios and Karim introduced promoted HEFT method improved by an iteration and parallel processing, which optimized simulation makespan(Kelefouras and Djemame, 2022). GA methods have been widely applied to task scheduling in heterogeneous systems. Hoseiny et al.'s priority-aware GA(Hoseiny et al., 2021) and Duan et al.'s improved GA with adaptable crossover and mutation rates(Duan et al., 2018) exemplify the single-objective optimization strategies deployed.

Moreover, the realm of multi-objective optimization (MO) in scheduling, particularly prevalent in heterogeneous simulation landscapes, calls for more intricate solutions. To address Dynamic Flexible Job Shop Scheduling (DFJSS), which is designed on heterogeneous system, GA is also widely used. Sang et al. proposed an improved optimization algorithm named NSGA-III-APEV based on NSGA-III(Sang et al., 2020), while Zhu et al. accelerated GA training with an efficient sample selection algorithm(Zhu et al., 2023). Additionally, Whitley et al. scheduled heterogeneous satellite systems through adapted task ordering strategies and improved genetic algorithm(Whitley et al., 2023). Further, The novel application of Q-learning by Zhang et al. to guide Particle Swarm Optimization (PSO) underscores the growing integration of reinforcement learning in MO scheduling challenges(Zhang et al., 2024). Despite these advancements, the critical consideration of simulation time synchronization in heterogeneous systems remains unaddressed, suggesting an significant gap for further exploration and innovation.

## 3 HDSP DEVELOPMENT

### 3.1 Architecture

Figure 1 illustrates HDSP's architecture, featuring a control center (root node) and multiple simulation nodes (sub-nodes), which engage in distributed communication via DDS middleware(eProsima, 2024). The distinction between control center and simulation nodes lies in D-MO Scheduler. Control center's D-MO Scheduler allocates simulation tasks to simulation nodes with genetic algorithms, while simulation nodes' schedulers only manage local tasks. Simulation nodes consist of the Simulation Module and Simulation-Script Interact Module. The functions of main components are described below:

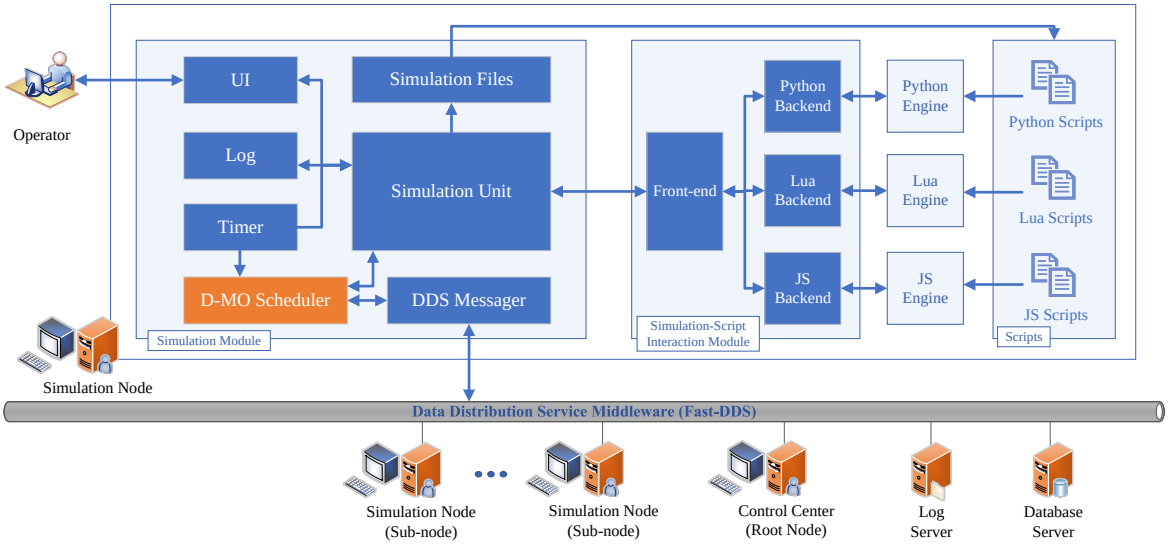


Figure 1: HDSP architecture.

1. Simulation Unit: Perform simulation calculations utilizing computational resources.
2. DDS Messenger: Communicates data and control information via Fast-DDS middle ware, an implementation of data-centric communication protocol.
3. D-MO Scheduler: Optimize and perform simulation task scheduling with GA. The scheduling algorithm satisfies simulation time synchronization constraints, and optimizes multiple objectives.
4. Simulation-Script Interact Module: Provides simulation APIs for different script languages and integrates multiple script engines. It also offers a uniform interface for calling scripts to the Simulation Module.

## 3.2 Simulation-Script Interaction Module

The Simulation-Script Interaction Module comprises of a front end for interface methods, and multiple back ends that manage script engines. This module integrates Python, Lua, and JavaScript engines through a unified interface, and connects simulation APIs to various script engines, allowing script-based simulation control. It supports AI script integration in Python and dynamic engine switching.

### 3.2.1 Script Operation Interface

The interaction layer is divided into two sub-layers: the front end, providing a unified script operation interface, and the back end, which operates the multiple script engines. The interface methods are defined in the front end, and implemented in the back end.

### 3.2.2 Data Type Conversion

The data type conversion in Simulation-Script Interaction Module is illustrated in Figure 2. This module facilitates compatibility between C++ and script languages including Python, Lua and JavaScript through intermediate data types.

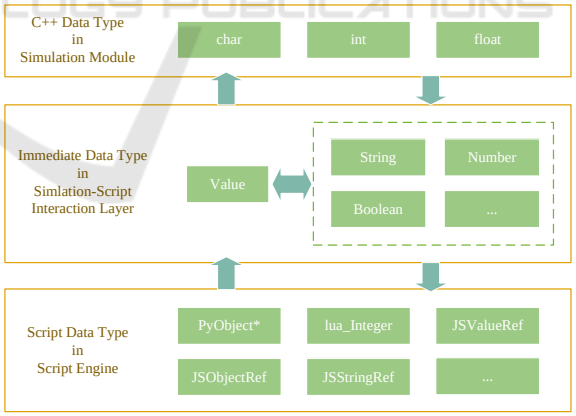


Figure 2: Data type conversion between C++ and script languages, including Python, Lua and JavaScript.

### 3.2.3 Simulation API Binding

The Simulation-Script Interaction Module binds simulation APIs to script engines, enabling scripts to monitor and control simulation process. Table 1 presents some supported simulation APIs.

Table 1: Part of simulation APIs for script engines.

| Function Name | Description |
|---|---|
| initialize() | Initialize simulation environment |
| configure(paras) | Configure simulation parameters |
| createEntity(args) | Create an entity |
| removeEntity(id) | Remove an entity |
| getEntityState(id) | Get the state of an entity |
| setEntityState(id, state) | Set the state of an entity |

# 4 D-MO SCHEDULER

## 4.1 System Model

Figure 3 presents the D-MO Scheduler structure, with Global Scheduler (GS) at the root node and Local Scheduler (LS) at sub-nodes. GS optimizes task scheduling policies via GA, while LS schedules tasks based on resources, timestamps, and simulation time synchronization constraints. The simulation time synchronization constraints in D-MO Scheduler are realized through the Lower Bound on Time Stamp (LBTS) and lookahead(Zhiwu and Yanfeng, 2009).

The workflow of D-MO Scheduler involves four phases: task allocation, task scheduling, simulation advancement, and synchronization.

During the task allocation phase, GS computes the global LBTS using equation 1, and notifies all LSs. Afterwards, based on the optimal allocation policies GS allocates simulation tasks to the corresponding simulation nodes. Assuming that there are $M$ tasks to be scheduled, the $LBTS_{Global}$ is determined as:

$$LBTS_{Global} = \min_i(TS_i + lookahead_i), i \in [1, M] \quad (1)$$

During task scheduling phase, each LS receives $LBTS_{Global}$ and the tasks list from the GS, then schedules the assigned tasks in parallel. The LS schedules a task immediately if its resource consumption falls within the remaining resources of the simulation node; otherwise, it waits for currently running tasks to complete and release resources before proceeding to schedule the next one.

During simulation advancement phase, each simulation node conducts computation.

Finally, during the synchronization phase, LS waits until the local simulation time $T_{local}$ reaches $LBTS_{Global}$, while recording the consumed wall-clock time of this round as $\Delta T_{wallclock}$. Subsequently, LS updates GS with the new tasks and $\Delta T_{wallclock}$, then GS initiates the next cycle, starting with task allocation.

## 4.2 Problem Formulation

We consider required data for scheduling as shown in Figure 3, which consists of $M$ simulation tasks, indexed by $m = \{1, 2, ..., M\}$, and $N$ simulation nodes, denoted by $n = \{1, 2, ..., N\}$. Each simulation task runs on a single simulation node, which can process multiple tasks concurrently.

**Definition 1 (Task):** $Task_m$ is defined by Eq.2, detailing its timestamp $TS_m$, execution time on the root node $T_{exe\_std\_m}$, priority weight $\lambda_i$ , and required computing resources $R_m = \{R_{m\_cpu}, R_{m\_gpu}, R_{m\_ram}\}$.

$$Task_m = \{TS_m, lookahead_m, T_{exe\_std\_m}, \lambda_m, R_m\} \quad (2)$$

**Definition 2 (Node):** A $Node_n$ is characterized by its computing resources $R_n = \{R_{n\_cpu}, R_{n\_gpu}, R_{n\_ram}\}$ and relative simulation rate $Rate_n$, as defined in Eq. 3.

$$Node_n = \{R_n, Rate_n\} \quad (3)$$

**Definition 3 (Relative Simulation Rate):** $Rate_n$ measures a node's processing speed, which is relative to the root node's rate ($Rate_{std}$).

$$Rate_n = \frac{Rate_{abs\_n}}{Rate_{std}} \quad (4)$$

$Rate_{abs\_n}$ represents the $Node_n$'s absolute simulation rate based on simulation step size $Step$, and elapsed wall clock time during one step $\Delta T_{wallclock\_n}$.

$$Rate_{abs\_n} = \frac{Step}{\Delta T_{wallclock\_n}} \quad (5)$$

**Definition 4 (Task Execution Time):** The execution time for $Task_m$ on $Node_n$ is given by:

$$T_{exe\_m} = \frac{1}{Rate_n} \cdot T_{exe\_std\_m} \quad (6)$$

**Definition 5 (Task Wait Time):** The wait time for $Task_m$ on $Node_n$ depends on the execution time of preceding task groups ($PEG$) and computed by Eq.7. $PEG_i$ constitutes a set of tasks that can be executed concurrently on a node, given as $PEG_i = \{Task_1, Task_2, ..., Task_{L_i}\}$.

$T_{wait\_m}$ accumulates the execution time of all $PEG$s before $Task_m$, with each $PEG_i$'s execution time being the longest standard execution time $T_{exe\_std\_j}$ among its tasks, adjusted by $Node_n$'s simulation rate.

$$\begin{aligned} T_{wait\_m} &= \sum_{i=1}^{W} T_{exe\_PEG_i} \\ &= \frac{1}{Rate_n} \cdot \sum_{i=1}^{W} \max_j(T_{exe\_std\_j}), j \in [1, L_i] \end{aligned} \quad (7)$$

**Definition 6 (Simulation Acceleration Ratio, SAR):** The Simulation Acceleration Ratio (SAR) evaluates scheduling efficiency, defined by the ratio of serial to distributed simulation wall-clock time:

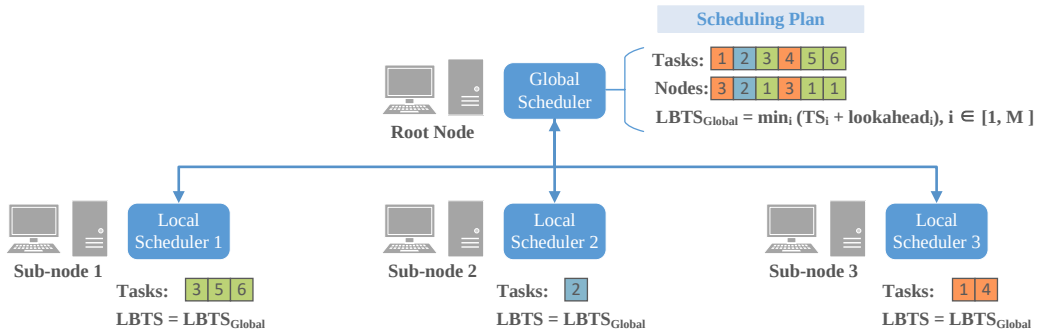


Figure 3: D-MO Scheduler.

$$SAR = \frac{T_{serial\_makespan}}{T_{distributed\_makespan}} \tag{8}$$

Serial simulation time, $T_{serial\_makespan}$, sums the standard execution time:

$$T_{serial\_makespan} = \sum_{m=1}^{M} T_{exe\_std\_m} \tag{9}$$

For a $Node_n$ at step $x$, where $k$ simulation tasks are completed, the distributed simulation time accounts for the maximum waiting and execution time:

$$T_{wallclock\_x\_n} = \max_i (T_{wait\_i} + T_{exe\_i}), i \in [1,k] \tag{10}$$

Each parallel cycle is signified by the interval between $LBTS$ updates. During the $i$-th parallel cycle, with simulation time $x \in [LBTS_{i-1}, LBTS_i)$, the total running time (wall-clock time) of $Node_n$ is:

$$T_{makespan\_n\_i} = \sum_{x=LBTS_{i-1}}^{LBTS_i} T_{wallclock\_x\_n} \tag{11}$$

The total time for distributed simulation accumulates over $S$ parallel cycles:

$$T_{parallel\_s} = \max_n (T_{makespan\_n\_s}), n \in [1,N] \tag{12}$$

Therefore, the total time consumption (wall-clock time) for distributed parallel simulation is:

$$T_{distributed\_makespan} = \sum_{s=1}^{S} T_{parallel\_s} \tag{13}$$

Consequently, the SAR is calculated as:

$$SAR = \frac{T_{serial\_makespan}}{T_{distributed\_makespan}} = \frac{\sum_{i=1}^{n} T_{exe\_std\_i}}{\sum_{s=1}^{S} \max_n (T_{makespan\_n\_s})} \tag{14}$$

### 4.3 Optimized Objectives

The D-MO Scheduler focuses on optimizing three objectives: *SAR* (Simulation Acceleration Ratio), *AWWT* (Average Weighted Waiting Time), and *RU* (Resource Utilization), aiming to enhance efficiency, fairness, and load balancing.

1. *SAR* measures the efficiency by comparing serial and distributed simulation time. Our proposed D-MO Scheduler aims to maximize the *SAR*:

$$SAR = \frac{\sum_{i=1}^{n} T_{exe\_std\_i}}{\sum_{s=1}^{S} \max_n (T_{makespan\_n\_s})} \tag{15}$$

2. The D-MO scheduler aims to minimize task waiting time by optimizing *AWWT* by the following minimization function:

$$AWWT = \frac{1}{M} \sum_{m}^{M} (\lambda_m \cdot T_{wait\_m}) \tag{16}$$

3. The D-MO Scheduler aims to maximize *RU* which is measure of the efficiency of resource:

$$RU = \frac{\sum_{m}^{M} (\frac{1}{Rate_m} \cdot R_m \cdot T_{exe\_m})}{\sum_{n}^{N} R_n \cdot T_{wallclock\_makespan}} \tag{17}$$

## 5 EXPERIMENTS AND RESULTS

### 5.1 Experimental Design

To assess the performance of MO genetic algorithm NSGA-III against the baseline algorithms (Random, Greedy, and Polling) in task scheduling, we conducted comparative experiments over 16 problem instances, focusing on three optimization goals: SAR, AWWT, and RU. We carried out 30 independent trials for both NSGA-III and Random algorithms. Each problem instance ranged from 50 to 600 tasks across 10 to 15 nodes. To introduce randomness of tasks and

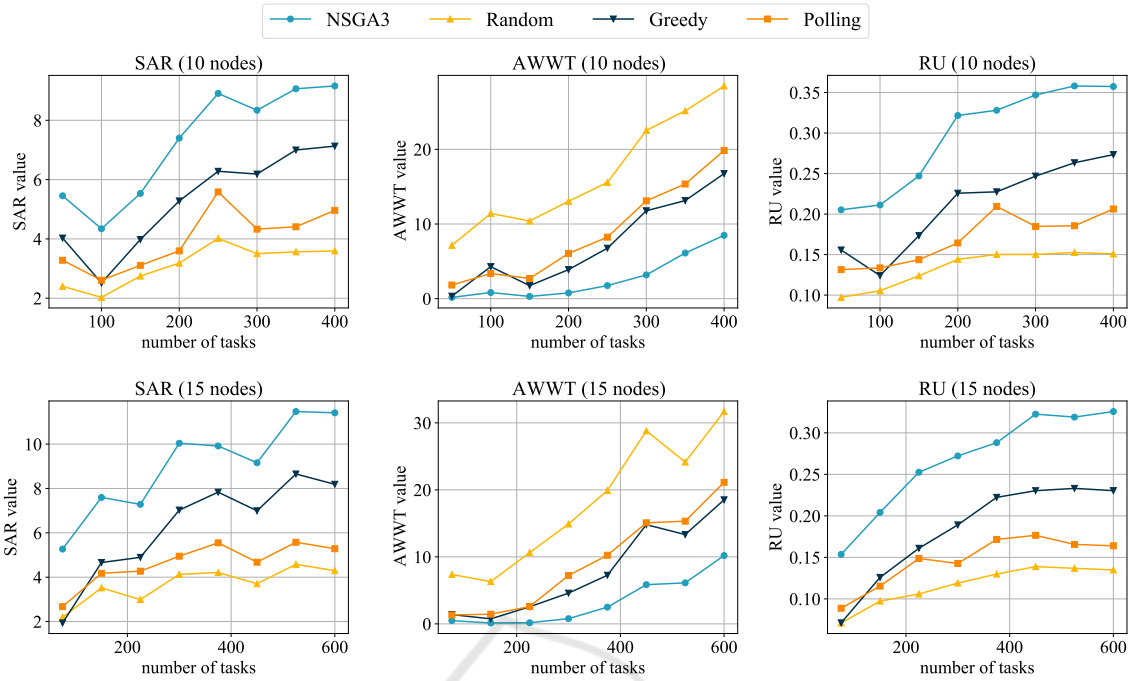


Figure 4: Curves of optimization objectives (SAR, AWWT, RU) of NSGA-III, Random, Greedy and Polling algorithms.

the heterogeneity of nodes, we randomly generate parameters such as priority and resources. Task time stamps were randomly assigned within 10 simulation steps. This setup aimed to form different node loads across different instances.

We employ NSGA-III as the multi-objective optimization genetic algorithm. Its parameters include a population size of 50, 500 evolution, a mutation probability of 0.001, a crossover probability of 0.8, and the use of binary coding with random initialization. NSGA-III's optimal solutions form the Pareto frontier, from which we compute SAR, AWWT, and RU for each solution. Subsequently, we average these values to derive experimental results.

To demonstrate the effectiveness of NSGA-III, we conducted comparative analysis with the Random, Greedy, and Polling algorithms, as shown in Figure.4. The Random algorithm distributes tasks uniformly across nodes, the Greedy algorithm prioritizes nodes with the most available resources, and the Polling algorithm allocates tasks to nodes in a sequential manner.

## 5.2 Scheduling Performance

Figure 4 illustrates the results of NSGA-III, Random, Greedy, and Polling algorithms across three objectives, with NSGA-III and Random curves averaged from 30 trials. Compared to baseline algorithms, NSGA-III significantly improves the efficiency of simulation scheduling, evidenced by higher SAR, lower AWWT, and increased RU. The experimental findings outlined in Table 2 further substantiates the advantages of NSGA-III over Random, Greedy, and Polling algorithms across all objectives, with these benefits particularly evident under heavier node loads. This validates NSGA-III's superior load tolerance and adaptability across 16 diverse problem instances.

Additionally, it's worth noting that Greedy slightly outperforms Polling, especially under high loads. This could attribute to Greedy's dynamic resource-based task allocation strategy. However, Greedy approach is susceptible to local optima, which is inferior to NSGA-III.

## 5.3 Distribution Analysis

Figure 5 shows SAR, AWWT, and RU distributions from 30 NSGA-III and Random algorithm trials. Regarding the distribution of optimization results, we observe that NSGA-III generates superior scheduling policies and more concentrated optimization outcomes, especially for AWWT. Table 3 provides standard deviation of SAR, AWWT, and RU from 30 NSGA-III and Random algorithm trials across 16 instances. NSGA-III's standard deviation is similar to Random's for SAR, but is consistently lower for AWWT (100% improvement) and RU (87.5% improvement). The results reveals that NSGA-III algorithm demonstrates superior distributional centraliza-

Table 2: The comparison results for algorithms NSGA-III, Random, Greedy and Polling.

| ins | NSGA-III | | | Random | | | Greedy | | | Polling | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | SAR | AWWT | RU | SAR | AWWT | RU | SAR | AWWT | RU | SAR | AWWT | RU |
| 1 | 5.45 | 0.17 | 2.05E-01 | 2.40 | 7.14 | 9.72E-02 | 4.03 | 0.33 | 1.56E-01 | 3.28 | 1.82 | 1.31E-01 |
| 2 | 4.34 | 0.82 | 2.11E-01 | 2.03 | 11.42 | 1.05E-01 | 2.52 | 4.28 | 1.24E-01 | 2.61 | 3.37 | 1.34E-01 |
| 3 | 5.53 | 0.30 | 2.47E-01 | 2.75 | 10.41 | 1.24E-01 | 3.98 | 1.74 | 1.73E-01 | 3.11 | 2.70 | 1.44E-01 |
| 4 | 7.40 | 0.76 | 3.22E-01 | 3.18 | 13.04 | 1.44E-01 | 5.28 | 3.89 | 2.26E-01 | 3.60 | 6.06 | 1.64E-01 |
| 5 | 8.91 | 1.75 | 3.28E-01 | 4.02 | 15.57 | 1.50E-01 | 6.28 | 6.76 | 2.27E-01 | 5.58 | 8.24 | 2.09E-01 |
| 6 | 8.34 | 3.18 | 3.47E-01 | 3.50 | 22.54 | 1.50E-01 | 6.19 | 11.77 | 2.47E-01 | 4.33 | 13.10 | 1.85E-01 |
| 7 | 9.06 | 6.11 | 3.58E-01 | 3.56 | 25.16 | 1.53E-01 | 7.00 | 13.14 | 2.63E-01 | 4.41 | 15.36 | 1.86E-01 |
| 8 | 9.16 | 8.48 | 3.57E-01 | 3.60 | 28.49 | 1.51E-01 | 7.13 | 16.75 | 2.73E-01 | 4.96 | 19.85 | 2.06E-01 |
| 9 | 5.27 | 0.48 | 1.54E-01 | 2.20 | 7.37 | 7.10E-02 | 1.94 | 1.39 | 7.11E-02 | 2.67 | 1.32 | 8.88E-02 |
| 10 | 7.60 | 0.14 | 2.04E-01 | 3.52 | 6.31 | 9.73E-02 | 4.66 | 0.75 | 1.26E-01 | 4.18 | 1.45 | 1.15E-01 |
| 11 | 7.28 | 0.17 | 2.53E-01 | 2.99 | 10.64 | 1.06E-01 | 4.89 | 2.55 | 1.61E-01 | 4.27 | 2.58 | 1.49E-01 |
| 12 | 10.04 | 0.78 | 2.72E-01 | 4.13 | 14.91 | 1.19E-01 | 7.02 | 4.59 | 1.89E-01 | 4.95 | 7.23 | 1.43E-01 |
| 13 | 9.92 | 2.50 | 2.88E-01 | 4.21 | 19.92 | 1.30E-01 | 7.83 | 7.26 | 2.22E-01 | 5.55 | 10.24 | 1.72E-01 |
| 14 | 9.16 | 5.85 | 3.23E-01 | 3.71 | 28.81 | 1.39E-01 | 6.99 | 14.81 | 2.30E-01 | 4.68 | 15.08 | 1.77E-01 |
| 15 | 11.47 | 6.12 | 3.19E-01 | 4.58 | 24.15 | 1.37E-01 | 8.65 | 13.33 | 2.33E-01 | 5.57 | 15.33 | 1.66E-01 |
| 16 | 11.41 | 10.20 | 3.26E-01 | 4.29 | 31.69 | 1.35E-01 | 8.19 | 18.51 | 2.30E-01 | 5.28 | 21.13 | 1.64E-01 |

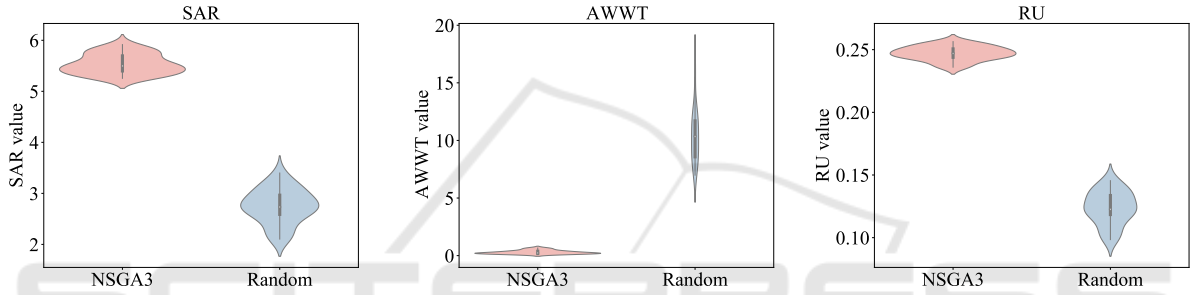


Figure 5: Violin plots of the SAR, AWWT, and RU distributions of the optimization results with NSGA-III and the Random algorithm.



tion and stability in optimization results, indicating NSGA-III's superior stability in optimization results.

Table 3: The standard deviation of the optimization objectives of NSGA-III and Random over 30 independent runs in 16 instances.

| ins | NSGA-III | | | RandomSolv | | |
|---|---|---|---|---|---|---|
| | SAR | AWWT | RU | SAR | AWWT | RU |
| 1 | 0.31 | 0.13 | 4.21E-03 | 0.39 | 2.52 | 1.30E-02 |
| 2 | 0.25 | 0.29 | 6.79E-03 | 0.21 | 2.90 | 9.24E-03 |
| 3 | 0.19 | 0.16 | 5.58E-03 | 0.33 | 2.33 | 1.32E-02 |
| 4 | 0.36 | 0.31 | 1.22E-02 | 0.33 | 1.68 | 1.29E-02 |
| 5 | 0.35 | 0.33 | 1.16E-02 | 0.36 | 1.88 | 1.21E-02 |
| 6 | 0.36 | 0.51 | 1.32E-02 | 0.27 | 2.62 | 1.03E-02 |
| 7 | 0.26 | 0.42 | 9.02E-03 | 0.30 | 2.40 | 1.22E-02 |
| 8 | 0.32 | 0.62 | 1.11E-02 | 0.30 | 2.19 | 1.16E-02 |
| 9 | 0.35 | 0.42 | 4.03E-03 | 0.29 | 2.78 | 6.82E-03 |
| 10 | 0.32 | 0.10 | 6.21E-03 | 0.31 | 1.30 | 7.69E-03 |
| 11 | 0.21 | 0.09 | 5.38E-03 | 0.30 | 2.14 | 8.78E-03 |
| 12 | 0.33 | 0.24 | 6.77E-03 | 0.46 | 2.15 | 1.11E-02 |
| 13 | 0.31 | 0.55 | 8.17E-03 | 0.37 | 2.24 | 1.04E-02 |
| 14 | 0.29 | 0.59 | 8.99E-03 | 0.35 | 2.34 | 1.14E-02 |
| 15 | 0.44 | 0.53 | 1.04E-02 | 0.38 | 2.15 | 1.03E-02 |
| 16 | 0.33 | 0.47 | 7.75E-03 | 0.35 | 2.47 | 9.49E-03 |

In summary, the D-MO simulation task scheduling problem is best solved by NSGA-III.

# 6 CONCLUSION

In this paper, we introduce the Heterogeneous Distributed Simulation Platform (HDSP) to facilitate Python-based AI algorithms, and the Dynamic Multi-Objective (D-MO) Scheduler supporting simulation time synchronization. HDSP integrates multiple script engines, improving its adaptability to diverse script languages. The NSGA-III algorithm enables the D-MO Scheduler to efficiently optimize key objectives: Simulation Acceleration Ratio (SAR), Average Weighted Waiting Time (AWWT), and Resource Utilization (RU). Our experiments demonstrate NSGA-III's superior efficacy, showing D-MO Scheduler's ability to outperform traditional scheduling methods. The statistical analysis also validates NSGA-III's load tolerance and distributional centralization, indicating its adaptability and reliability to diverse heterogeneous simulation configurations.

In future research, we will focus on integrating more advanced AI algorithms and expanding HDSP's support for emerging script languages, in order to meet the dynamic needs of both research and industrial applications.

## ACKNOWLEDGEMENTS

In this work, artificial intelligence (AI) was used for grammar checking and sentence optimization.

## REFERENCES

Adday, G. H., Subramaniam, S. K., Zukarnain, Z. A., and Samian, N. (2024). Investigating and analyzing simulation tools of wireless sensor networks: A comprehensive survey. *IEEE Access*, 12:22938–22977.

Chambon, C., Piller, O., and Mortazavi, I. (2024). A robust simulator of pressure-dependent consumption in python. *Journal of Hydroinformatics*, pages 284–303.

Chang, Y (Chang, Y., Wei, DX (Wei, D., Jia, HH (Jia, H., Curreli, C (Curreli, C., Wu, ZZ (Wu, Z., Sheng, M (Sheng, M., Glaser, SJ (Glaser, S. J., and Yang, XD (Yang, X. (2019). Spin-scenario: A flexible scripting environment for realistic mr simulations. *Journal of Magnetic Resonance*, pages 1–9.

D'Aquino, Z., Arabas, S., Curtis, J. H., Vaishnav, A., Riemer, N., and West, M. (2024). Pypartmc: A pythonic interface to a particle-resolved, monte carlo aerosol simulation framework. *SoftwareX*, 25:101613.

Deb, K. and Jain, H. (2014). An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. *IEEE Transactions on Evolutionary Computation*, 18(4):577–601.

Duan, K., Fong, S., Siu, S. W. I., Song, W., and Guan, S. S. (2018). Adaptive incremental genetic algorithm for task scheduling in cloud environments. *Symmetry*, 10(5):168.

eProsima (2024). Fast dds documentation. Accessed: 2024-03-26.

Hoseiny, F., Azizi, S., Shojafar, M., Ahmadiazar, F., and Tafazolli, R. (2021). Pga: A priority-aware genetic algorithm for task scheduling in heterogeneous fog-cloud computing. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6.

Ierusalimschy, R., De Figueiredo, L. H., and Celes, W. (2011). Passing a language through the eye of a needle. *Communications of the ACM*, 54(7):38–43.

Jawaddi, S. N. A. and Ismail, A. (2024). Integrating openai gym and cloudsim plus: A simulation environment for drl agent training in energy-driven cloud scaling. *Simulation Modelling Practice and Theory*, page 102858.

Kelefouras, V. I. and Djemame, K. (2022). Workflow simulation and multi-threading aware task scheduling for heterogeneous computing. *J. Parallel Distributed Comput.*, 168:17–32.

Liu, Z., Chen, Z., and Song, K. (2021). Spinspj: a novel nmr scripting system to implement artificial intelligence and advanced applications. *BMC bioinformatics*, page 581.

Lu, C., Gao, L., Yi, J., and Li, X. (2020). Energy-efficient scheduling of distributed flow shop with heterogeneous factories: A real-world case from automobile industry in china. *IEEE Transactions on Industrial Informatics*, 17(10):6687–6696.

Sang, Y., Tan, J., and Liu, W. (2020). Research on many-objective flexible job shop intelligent scheduling problem based on improved nsga-iii. *IEEE Access*, 8:157676–157690.

Souza, P., Ferreto, T., and Calheiros, R. N. (2023). Edgesimpy: Python-based modeling and simulation of edge computing resource management policies. *Future Gener. Comput. Syst.*, 148:446–459.

Whitley, D., Quevedo De Carvalho, O., Roberts, M., Shetty, V., and Jampathom, P. (2023). Scheduling multi-resource satellites using genetic algorithms and permutation based representations. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1473–1481.

Wong, T., Timoshkin, I. V., Macgregor, S. J., Wilson, M. P., and Given, M. J. (2023). The design of a python library for the automatic definition and simulation of transient ionization fronts. *IEEE Access*, 11:26577–26592.

Wornow, M., Gyang Ross, E., Callahan, A., and Shah, N. H. (2023). Aplus: A python library for usefulness simulations of machine learning models in healthcare. *Journal of Biomedical Informatics*, 139:104319.

Zhang, W., Li, C., Gen, M., Yang, W., and Zhang, G. (2024). A multiobjective memetic algorithm with particle swarm optimization and q-learning-based local search for energy-efficient distributed heterogeneous hybrid flow-shop scheduling problem. *Expert Syst. Appl.*, 237(Part C):121570.

Zhiwu, D. and Yanfeng, L. (2009). Dynamic time management algorithms research in simulation system hla-based. In *2009 Second International Workshop on Computer Science and Engineering*, volume 1, pages 580–583.

Zhu, L., Zhang, F., Zhu, X., Chen, K., and Zhang, M. (2023). Sample-aware surrogate-assisted genetic programming for scheduling heuristics learning in dynamic flexible job shop scheduling. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 384–392.