Task scheduling optimization problem (offloading) Multi-objective optimization

BARBOSA Mathias, THIRÉ Mael

ING3 IA1

2023 / 2024

Contents

1 Introduction

In the class of Multi-objective optimization, we are assigned a project divided in three phases:

First, we need to formulate an optimization problem based on Cloud Computing field. Then, we are asked to solve it with given data containg three cloud machines and ten fog machines, using a metaheuristic function based on a pareto approach. Finally, we have to evaluate the performances and compare our results with other groups.

2 Problem and mathematical formulation

Problem: Task Scheduling in Fog-Cloud Computing Environments

Objective: Assign tasks to various virtual machines (VMs) efficiently i.e minimizing the total cost and the makespan. We initially thought we could try to minimize the energy consumption but because of the lack of data, we will only focus on the two others objectives.

Decision variables: X_{ij}: Indicates whether task i is assigned to VM j i.e. $X_{ij} = 1$ if task i is assigned to VM j and $X_{ij} = 0$ otherwise.

Constraints:

1. Task Assignment Constraint: Each task should be assigned to exactly one VM.

   $\Sigma_j X_{ij} = 1, \forall i$

2. VM Capacity Constraint: The total workload assigned to each VM should not exceed its capacity.

   $\Sigma_i W_i \cdot X_{ij} \leq C_j, \forall j$

   where W_i is the workload of task i i.e. the number of instructions needed in millions, and C_j is the process capacity of VM j corresponding to CPU rate of VM j in 10s.

Objective functions:

1. Minimize Makespan: Minimize the total time taken to complete all tasks, i.e., the makespan.

   Minimize $\Sigma_i \Sigma_j T_{ij} \cdot X_{ij}$

   where T_{ij} is the time taken to complete task i on VM j. For clarity purpose, T_{ij} is expressed in milliseconds.

   $T_{ij} = \frac{W_i}{W_{j,ps}}$

   where $W_{j,ps}$ is the number of instructions in millions, per second of VM j, i.e. CPU rate.

2. Minimize Cost: Minimize the total cost of VMs.

   Minimize $\Sigma_i \Sigma_j (c_{ijp} + c_{ijm} + c_{ijb}) * X_{ij}$

   where c_{ijp} is the CPU usage cost needed for task i using VM j, c_{ijm} is the memory usage cost needed for task i using VM j and c_{ijb} is the bandwidth usage needed for task i using VM j.

These objective functions represent different optimization goals for work scheduling in a cloud computing environment. The second one minimizes cost consumption, contributing to cost-efficiency.

3 Metaheuristic Function

In our project, we employed the multi-objective bat algorithm to address the task scheduling optimization problem in fog-cloud computing environments. The algorithm operates by simulating the foraging behavior of bats in the dark. Here's how the algorithm works:

1. **Population Initialization**: An initial population of bats is generated, with each bat representing a potential solution to the task scheduling problem via its position: the priority order of the tasks. Parameters such as the number of bats, problem dimensions, and algorithmic parameters are defined. In our implementation, we read task details, node details, execution table, and cost table from an Excel file to initialize the problem parameters. These include the number of tasks, the number of virtual machines, task workloads, VM capacities corresponding to the CPU rate for 100ms, and various costs associated with VM usage.

2. **Search and Solution Update**: At each iteration of the algorithm, each bat performs a local search around its current position. This local search is guided by the quality of a best solution i.e. the last solution in the archive, each bat not present in the archive, updates its position according to the best bat position and its pulsation rate increases a little bit. If the pulsation rate of the bat is not loud enough, the bat position becomes very close to the best bat position.

3. **Termination Criterion**: The algorithm continues iterating until a predefined termination criterion is met. In our implementation, we set a maximum number of epochs (iterations) as the termination criterion. This ensures that the algorithm terminates after a specified number of generations, providing a balance between computational resources and solution quality. The termination criterion can be adapted based on problem-specific requirements or computational constraints, ensuring flexibility in algorithm execution.

4. **Analysis of Solutions**: Once the algorithm completes, the non-dominated solutions, i.e., those not outperformed by another solution in all objectives, are analyzed. These solutions represent the Pareto front and offer a trade-off between the different objectives of the problem. Analyzing these solutions provides insights into the inherent trade-offs and enables decision-makers to select solutions that align with their preferences or constraints. Decision-making techniques, such as sensitivity analysis or preference articulation, can be employed to further refine the set of non-dominated solutions and identify the most suitable solution based on the specific needs of the problem.

In summary, the multi-objective bat algorithm is an effective method for solving complex optimization problems by finding a set of solutions that represent a compromise between multiple competing objectives. For the implementation details of the algorithm, please refer to the code provided in the appendix.

4 Performances and comparison

4.1 Performances

The performance evaluation of the multi-objective bat algorithm is essential to assess its effectiveness in solving the task scheduling optimization problem in fog-cloud computing environments. In this section, we present the performance metrics and results obtained from the algorithm execution.

The non-dominated solutions obtained from the algorithm execution consist of Makespan and Total Cost for each solution.

Our algorithm is in $O(enm)$ with e the number of epochs, n the number of tasks and m the number of bats. So, considering a fixed e and a fixed m, the execution time is linear in n, which is excellent. To improve it, we would have to find an algorithm in $O(ln(n))$ which may be very difficult to find. Hence, we are satisfied, in fact: in just 30 seconds, it can solve a task scheduling problem with 40 tasks in 100 epochs with a population of 100 bats. If we look at the plot Figure 5, we can see the convergence of the algorithm. The results look great and we can see the evolution of the bats. For the same amount of time but with 120 tasks and only 30 bats in compensation, it also converges to a sufficient Pareto front (cf Figure 7), but here, they stayed relatively close to each other.

In summary, the multi-objective bat algorithm demonstrates promising performances in addressing the task scheduling optimization problem, as evidenced by the obtained non-dominated solutions.

4.2 Comparison

We decided to compare with the solution of the group of Justine and David. They solved a task scheduling problem with a genetic algorithm but with different constraints. Indeed, they chose to use a memory capacity constraint rather than a CPU capacity like us. They also chose a different calculation for the makespan. Our results are therefore not completely comparable but we ended up with the same execution time, so they are both performant and give solutions that are graphically sufficient.

5 Conclusion

In conclusion, our study on task scheduling optimization in fog-cloud computing environments has yielded promising results. By employing the multi-objective bat algorithm, we effectively tackled the complex problem of task assignment while minimizing both execution time and associated costs.

The solutions obtained were analyzed through the lens of the Pareto frontier, providing a trade-off between competing objectives of operational efficiency and cost-effectiveness. We demonstrated that our approach enabled the discovery of a set of efficient solutions, where no solution was dominated by another across all considered objectives.

Furthermore, the exploration of the search space allowed for the identification of diverse solutions, contributing to the robustness and versatility of our approach. The visualization of the Pareto front provided insights into the trade-offs inherent in the task scheduling problem, guiding decision-making towards optimal solutions.

Overall, our study showcases the efficacy of the multi-objective bat algorithm in addressing real-world optimization challenges in fog-cloud computing environments. Our findings pave the way for enhanced resource utilization, improved system performance, and cost-efficient operations in cloud-based systems.

As a perspective of improvement, we could complete the database with power measures, in order to get the energy consumption of the VMs as a fitness function, like initially wanted. It would make the results more interesting in terms of environnement sustainability.

1 Implementation Details

```
# Archive part
for bat in population:
    # Get and evaluate a solution depending on position of the bat
    solution = bat_task_assignment(bat.pos, workload, capacity)
    bat_fitness = [makespan(solution), total_cost(solution)]
    bat.set_fitness(bat_fitness)
    bat.set_sol(solution)
    if bat not in archive:
        is_dominated = False
        for archived_bat in archive:
            if bat_fitness[0] < archived_bat.fitness[0] and bat_fitness[1] < archived_bat.fitness[1]:
                # If the new bat dominates an archived bat, remove the archived bat
                archive.remove(archived_bat)
            elif bat_fitness[0] >= archived_bat.fitness[0] and bat_fitness[1] >= archived_bat.fitness[1]:
                # If an archived bat dominates the new bat, it is dominated
                is_dominated = True

        if not is_dominated:
            # If the new bat is not dominated, add it to the archive
            archive.append(bat)
            # Selecting the last best bat as a reference
            best_bat = bat
    else:
      # Checking the dominance in the archive
      for archived_bat in archive:
        if bat.fitness[0]>=archived_bat.fitness[0] and bat.fitness[1]>=archived_bat.fitness[1] and bat.fitness!=archived_bat.fitness:
          archive.remove(bat)
    makespan_cost_epoch.append(bat_fitness)

makespan_costs.append(makespan_cost_epoch)
```

Figure 1: Python Code archive

```python
# Moving part
for bat in population:
  if bat not in archive:
    new_pos = [0.00 for _ in range(dim)]
    new_vel = [0.00 for _ in range(dim)]

    # Frequence of the bat
    freq = random.uniform(qmin, qmax)

    # Random pulsation
    pulse_chance = random.uniform(0, 1)
    for d in range(dim):
        # Updating velocity
        new_vel[d] = bat.vel[d] + (bat.pos[d] - best_bat.pos[d]) * freq

        # If pulsation rate not loud enough
        if pulse_chance > bat.pulse_rate:
            # Stays close to the best bat
            new_pos[d] = best_bat.pos[d] + (random.uniform(-1, 1) * 0.1)
        else:
            # Moving toward the best bat and keeps exploring
            new_pos[d] = bat.pos[d] + new_vel[d]

    # Update
    bat.vel = new_vel
    bat.pos = new_pos
    bat.pulse_rate = bat.max_pulse_rate * (1 - exp(-gam * e))
```

Figure 2: Python Code bat update

```python
def makespan(solution):
    completion_times = np.zeros(num_vms)
    max_completion_time = 0
    for i in range(num_vms):
        for j in range(num_tasks):
            completion_times[i] += times[i, j] * solution[i, j]
    return sum(completion_times)
```

Figure 3: Makespan

```python
def total_cost(solution):
    cost = 0
    for i in range(num_vms):
        for j in range(num_tasks):
            cost += costs[i, j] * solution[i, j]
    return cost
```

Figure 4: Cost

```
Non-dominated solutions in the archive:
Makespan: 852.1709999999999 Total Cost: 728.9665999999999
Makespan: 852.5653 Total Cost: 721.3927
Makespan: 862.6985 Total Cost: 697.5913000000002
Makespan: 854.8039 Total Cost: 708.4953000000002
Makespan: 861.2698999999999 Total Cost: 697.8824999999999
Makespan: 875.1215 Total Cost: 685.0234999999998
Makespan: 889.1443999999999 Total Cost: 679.2878999999998
Makespan: 866.8213999999999 Total Cost: 694.0148999999998
Makespan: 871.3139999999999 Total Cost: 687.0785
Makespan: 875.4606999999999 Total Cost: 683.8989999999998
Makespan: 874.8663999999999 Total Cost: 685.4075999999998
Makespan: 864.7931 Total Cost: 694.5058
Makespan: 859.423 Total Cost: 698.7812
Makespan: 868.6829 Total Cost: 687.5547999999999
Makespan: 858.2202 Total Cost: 702.6360000000001
Makespan: 864.2863 Total Cost: 694.5090999999999
Makespan: 861.7553 Total Cost: 697.7140999999998
Makespan: 868.0441 Total Cost: 691.8956999999999
Makespan: 868.4855 Total Cost: 691.2866999999998
Makespan: 868.0065 Total Cost: 693.7223
```

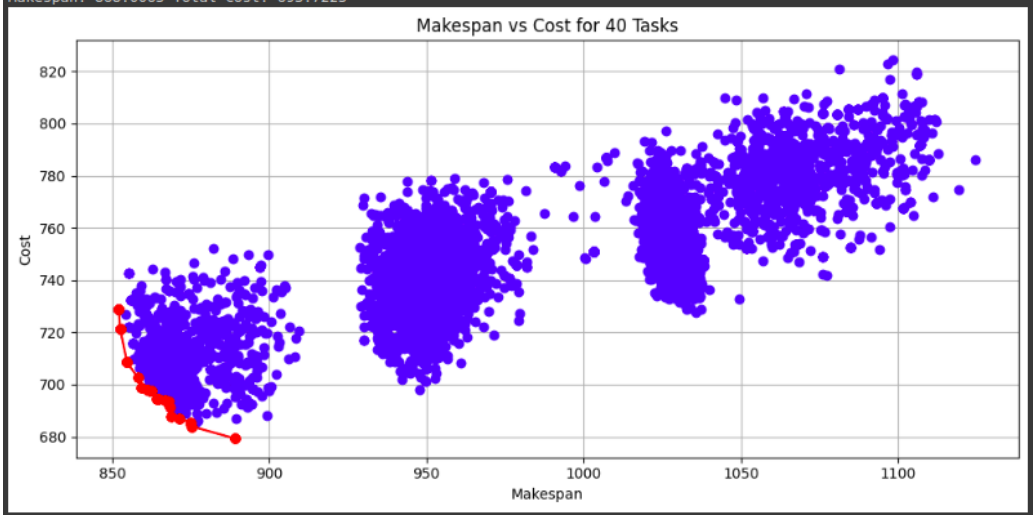

Figure 5: Pareto Front with 40 tasks, 100 epochs and population of 100 bats

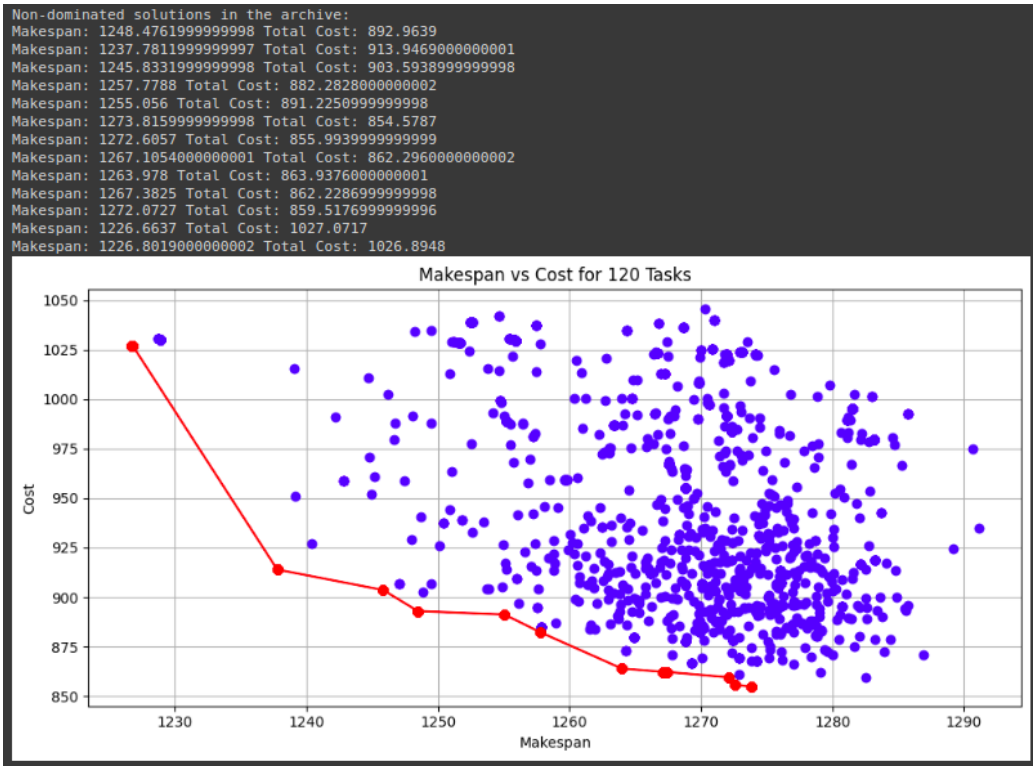

Figure 6: Pareto Front with 120 tasks, 100 epochs and population of 30 bats