Virtual Resource Allocation in MEC Using Matching Algorithms

Quantitative Evaluation of Stochastic Models

Gregorio Piqué

May 19, 2025

UNIVERSITÀ DEGLI STUDI FIRENZE

Table of Contents

Project Description

investigation of matching algorithms for VM-to-PM allocation in **Mobile Edge Computing (MEC)**

Goal: evaluate how different algorithmic strategies impact *energy consumption* and *resource utilization*

system model and general problem configuration setup based on [1]

Project's Approach: compare alternative matching strategies

- baseline: *Random*
- greedy: *First-Fit*, *Round-Robin*
- preferences/bids based: *Gale-Shapley (dynamic preferences)*, *Auction-based*

Table of Contents

2 Problem Formulation

System Model

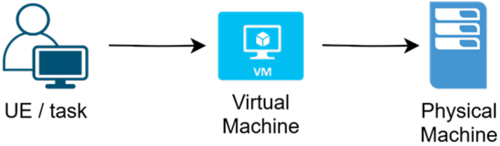
System Components

- UEs: generate tasks (with CPU/memory requirements)
- VMs: execute tasks; consume resources
- PMs: host VMs; have finite cores and memory

Allocation Flow

- task $\rightarrow$ VM (predefined)
- VM $\rightarrow$ PM (to be optimized)

Constraints: respect CPU/memory limits of VMs and PMs



Energy Consumption Model

energy consumption of MEC system depends on remote computation (E^{vm}: PM processing on MEC)

$$E^{vm} = \sum_{i=1}^{N} \sum_{j=1}^{M} (T - \tau)\theta b_{i,j} \alpha_{i,j} \xi_i$$

Details

- T: total duration
- τ: offloading phase duration
- θ: computing capability of a single PM core
- $b_{i,j}$: if VM vm_i is on PM pm_j
- $\alpha_{i,j}$: number of cores from vm_i on pm_j
- ξ_i: energy per processor cycle of vm_i

Objective

Goal

- minimize total energy consumption
- ensure valid VM-to-PM placement under resource constraints

Approach

- alternative to hypergraph-based optimization from original work
- use matching algorithms for VM-to-PM allocation
- evaluate performance in terms of energy efficiency and resource utilization

Table of Contents

3 Matching Algorithms

Random Algorithm (Baseline)

Overview

- baseline method for comparison
- VMs (pre-mapped to UE tasks) assigned randomly to PMs

Allocation Process

- random PM selected for each VM
- if insufficient resources, another PM is sampled
- repeats until feasible placement or no PMs available

Greedy Algorithm (First-Fit)

Overview

- greedy strategy for VM-to-PM allocation
- simple implementation
- VMs placed on first available PM

Allocation Process

- PMs evaluated in fixed order
- VM assigned to first PM with sufficient resources
- if no PM can accommodate, task remains unallocated
- PM list re-evaluated from beginning for each task

Round-Robin Algorithm

Overview

- variation of Greedy (First-Fit) strategy
- reduces PM hotspotting

Allocation Process

- PMs checked in circular order
- after assigning to i-th PM, next allocation attempt starts from $(i + 1)$-th PM
- continues cycling through PM list for each task

Gale-Shapley Algorithm — Overview

Concept

- based on Hospitals / Residents [2] variant of stable marriage problem
- matches VMs to PMs using dynamically computed preferences

Algorithm steps

1. VMs and PMs compute preference lists
2. unassigned VMs propose to preferred PMs
3. PMs tentatively accept the best proposals, rejecting others
4. rejected VMs update their list and propose again
5. repeat until all VMs are matched or no valid PMs remain

Goal

- achieve energy-efficient and resource-aware allocation

GS Preference Scoring — VM Perspective

VM preferences aim

- minimize energy consumption
- promote load balancing (prefer underutilized or empty PMs)

Preference Function

$$\text{preference}(vm_i, pm_j) = -\lambda \cdot E^{vm}(vm_i, pm_j) + \gamma \cdot \text{availableResources}(pm_j)$$

Details

- $E^{vm}(vm_i, pm_j)$: estimated energy cost of placing vm_i on pm_j
- all components are normalized to ensure comparability
- only feasible matches are considered
- λ: energy weight — higher values bias toward energy efficiency
- γ: load balancing weight — higher values bias toward more task-balanced solutions

GS Preference Scoring — PM Perspective

PM preferences aim

- reduce energy usage
- promote task consolidation (reduce resource fragmentation)

Preference Function

$$\begin{aligned}\text{preference}(pm_j, vm_i) = &- \lambda \cdot E^{vm}(vm_i, pm_j) \\ &+ \mu \cdot (\text{usedResources}(pm_j) + \text{requiredResources}(vm_i))\end{aligned}$$

Details

- consolidation term favors VMs that better utilize PM resources
- μ: consolidation weight — higher values to avoid PM underutilization

Note: consolidation can be enforced by proposers (VMs) by introducing the term in their preference scoring

Auction-Based Algorithm

Overview

- market-based method: VMs act as bidders, PMs as auctioned items
- iterative bidding process based on evaluation scores
- dynamic pricing encourages balanced and energy-efficient assignments

Evaluation Score

- each VM scores eligible PMs using:

$$\text{eval}_{vm_i}(pm_j) = -\lambda E_{vm} - \phi \cdot \text{price} - \gamma \cdot \text{loadFactor} + \rho \cdot \text{computeSpeed}$$

- encourages:
  - **low energy usage** and **low pricing** ($-\lambda\cdot$energy, $-\phi\cdot$price)
  - **load balancing** ($-\gamma\cdot$ load factor)
  - **PM performance** ($+\rho\cdot$ compute speed)
- all terms are normalized and weighted to tune behavior

Auction-Based Algorithm (continued)

Bidding & Matching Process

1. VMs evaluate PMs with $eval_{vm_i}(\cdot)$ function
2. unassigned VMs bid for the PM with highest score; bid from vm_i computed as:

$$\text{bid}_{vm_i} = \text{bestPMscore} - \text{secondBestPMscore} + \varepsilon$$

3. each PM accepts highest bid (bid*) rejecting others
4. PM price updated via EWMA:

$$\text{price}'_{pm_j} = \alpha \cdot \text{bid}^* + (1 - \alpha) \cdot \text{price}_{pm_j}$$

5. repeat until all VMs are matched or no valid PMs remain

Behavior

- dynamic pricing discourages oversubscription
- promotes both **efficiency** and **load-aware** distribution

Table of Contents

4 Implementation

Implementation

- project implemented in **Java**
- object-oriented design
- source code publicly available [3]

MEC System
4 Implementation

MecSystem

- userEquipments: List<UE>
- virtualMachines: List<VM>
- physicalMachines: List<PM>
- ue2VmMappings: List<Ue2VmMapping>
- totalDurationTime: double
- offloadingDurationTime: double

+ addUE(UE): void
+ addVM(VM): void
+ addPM(PM): void

MecMapping

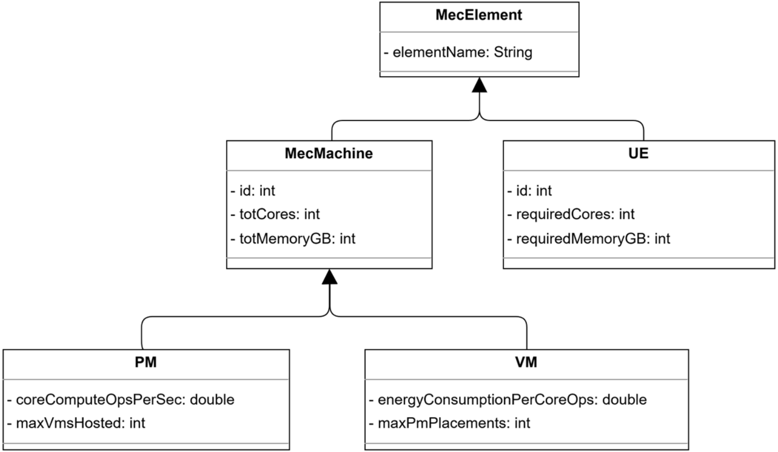
- vm2PmPlacement: List<List<Integer>>
- vmGb2PmPlacement: List<List<Integer>>
- vmCores2PmPlacement: List<List<Integer>>

+ setVmPlacement(int, int): void
+ removeVmPlacement(int, int): void

+ setVmCores2Pm(int, int, int): void
+ setVmGb2Pm(int, int, int): void

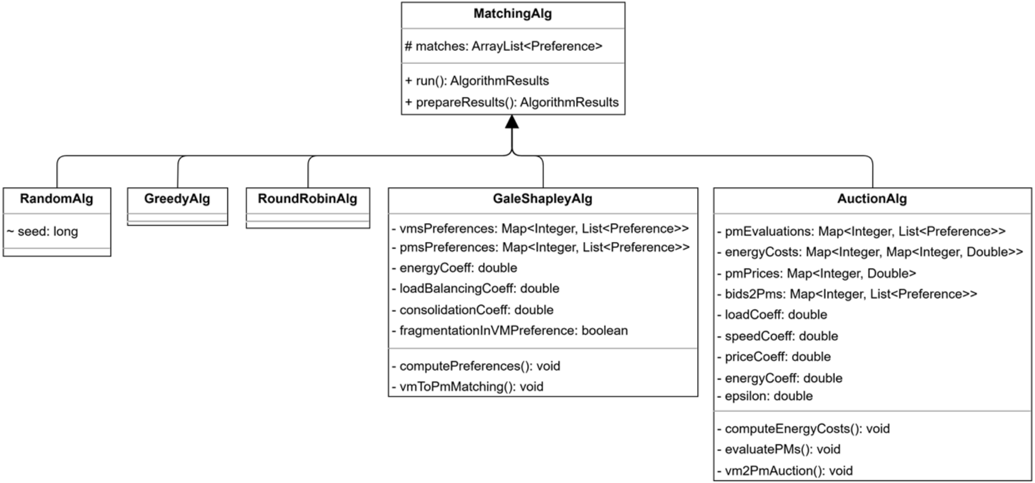
MEC System Elements

4 Implementation



Algorithms

4 Implementation



Algorithms' helper classes

4 Implementation

Ue2VmMapping

- ueId: int
- vmId: int
- cores: int
- memory: int

Preference

- proposerId: int
- receiverId: int
- preference: double
- ue2VmMapping: Ue2VmMapping

ResourceAvailability

- id: int
- totCores: int
- totMemory: int
- totAllocations: int
- usedCores: int
- usedMemory: int
- usedAllocations: int

+ areResourcesAvailable(int, int): boolean
+ allocateResources(int, int): void
+ releaseResources(int, int): void

<<record>>
AlgorithmResults

- algorithmName: String
- totalAllocatedUes: int
- totalAllocatedVms: int
- totalAllocatedPms: int
- totalEnergyConsumed: double

Services

4 Implementation

MecSystemService

+ checkAssignmentAllowed(int, int): boolean

+ addVMResourcesOnPm(int, int, int, int): void
+ removeVMResourcesOnPm(int, int, int, int): void

+ getVmsHostedByPm(int): List<Integer>
+ getPmsHostingVm(int): List<Integer>

+ getHostedVmsCoresByPm(int): List<Integer>
+ getHostedVmsGbsByPm(int): List<Integer>

+ getRemainingCoresInVm(int): int
+ getRemainingGbsInVm(int): int
+ getRemainingCoresInPm(int): int
+ getRemainingGbsInPm(int): int

EnergyConsumptionService

+ getEnergyConsumptionWithVmAndPm(VM, PM): double

+ getEnergyConsumptionPerVm(int): double
+ getEnergyConsumptionPerPm(int): double

+ getTotalEnergyConsumption(): double

Table of Contents

5 Experimental Tests

Setup and Goal

Configuration

- fixed number of PMs, VMs, and UEs
- PM/VM/UE resources randomly assigned (CPU, memory)
- randomization to ensure diverse and balanced scenarios

Objectives

1. compare all methods on energy use, execution time, and general allocation efficiency
2. study Gale-Shapley with different consolidation enforcer (proposer vs receiver)

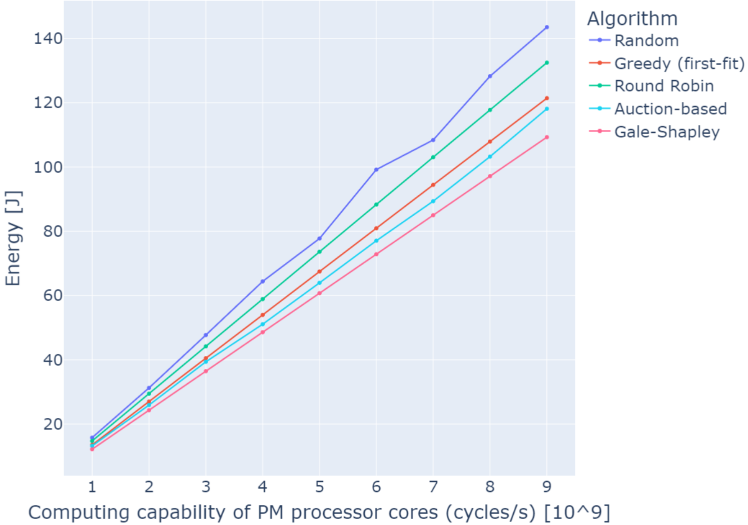
Algorithm Comparison

Objective: compare all methods on different evaluation criteria

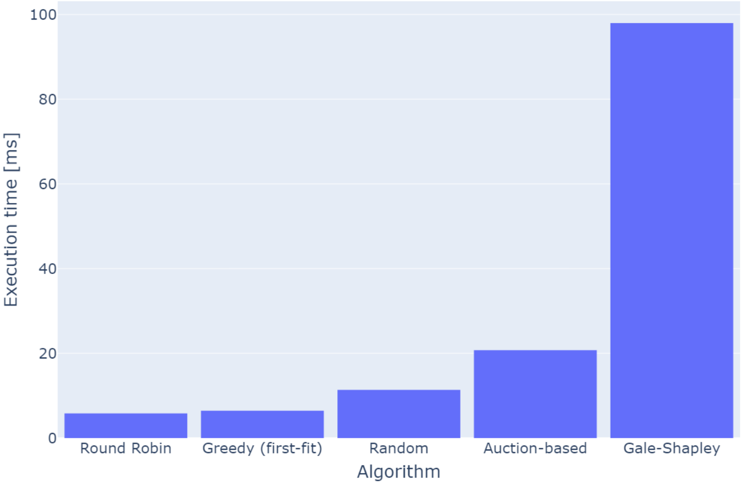
Tested Metrics

- energy consumption
- execution time
- general allocation efficiency

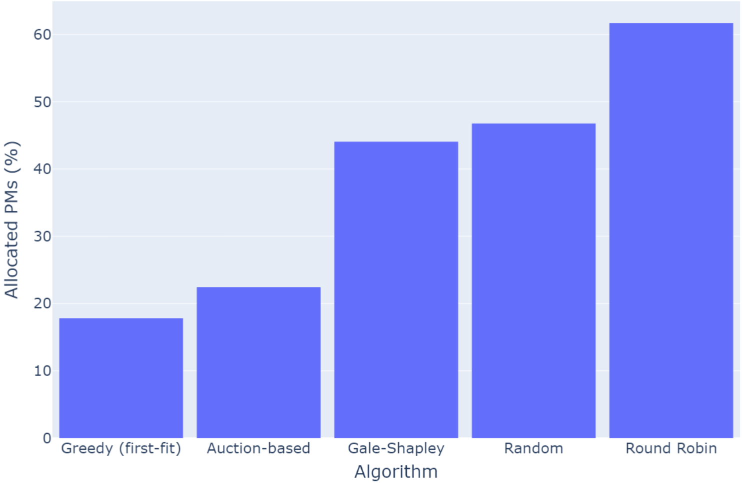
Energy Consumption vs. PM Computing Capabilities



Execution Time vs. Algorithm

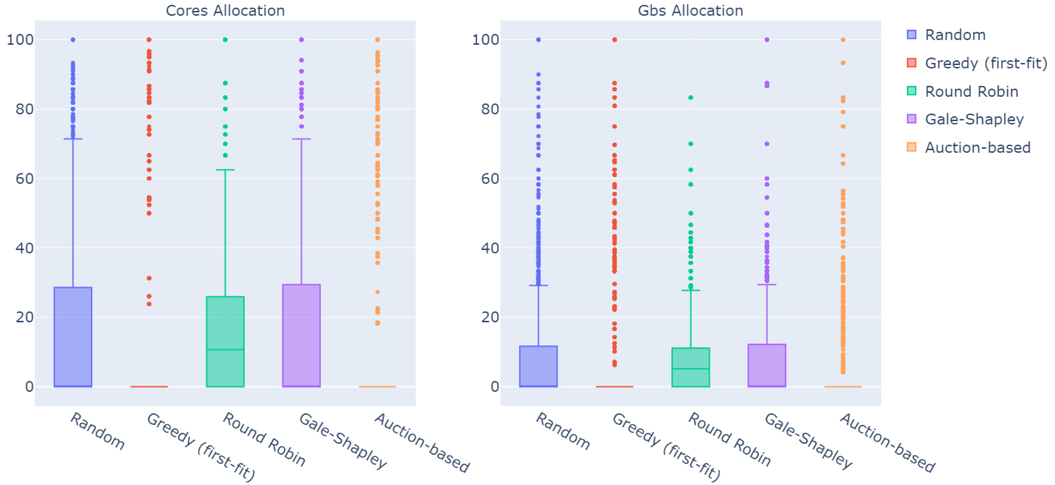


PM Allocation vs. Algorithm



PM Resource Utilization vs. Algorithm

5 Experimental Tests



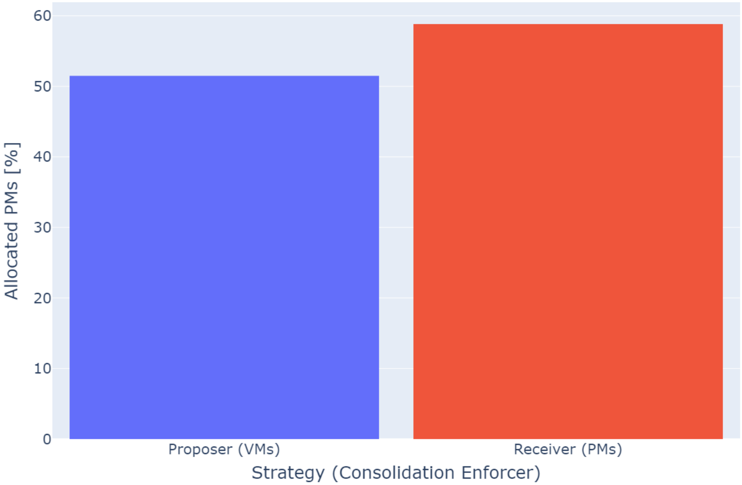
Gale-Shapley: Impact of Consolidation Enforcer

Objective: analyze how results change when the consolidation term is used by different roles in preference scoring

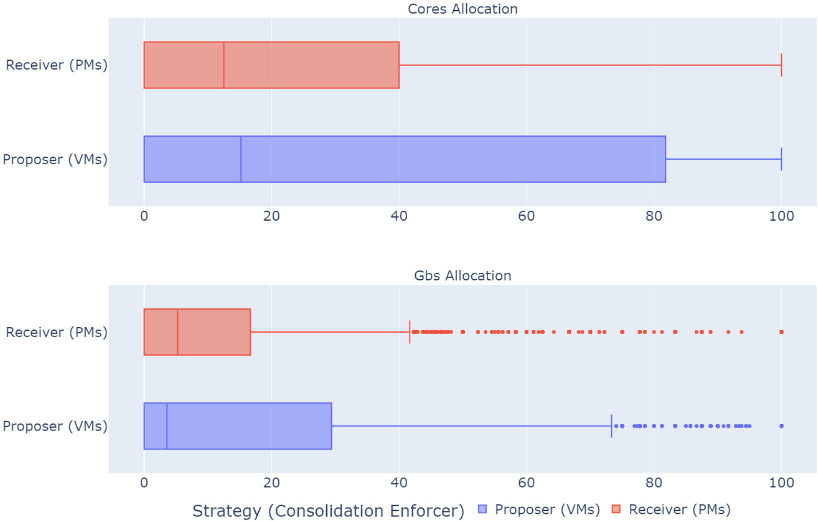
Tested Configurations

- **proposer-side (VMs)**: favors resource consolidation when bidding
- **receiver-side (PMs)**: promotes load-aware acceptance

PM Allocation vs. Consolidation Strategy

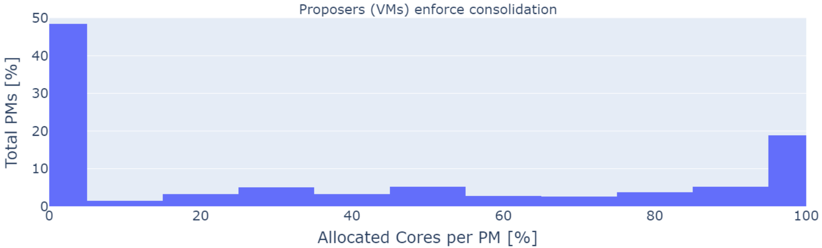


PM Resource Utilization vs. Consolidation Strategy



PM Core Utilization vs. Consolidation Strategy

5 Experimental Tests



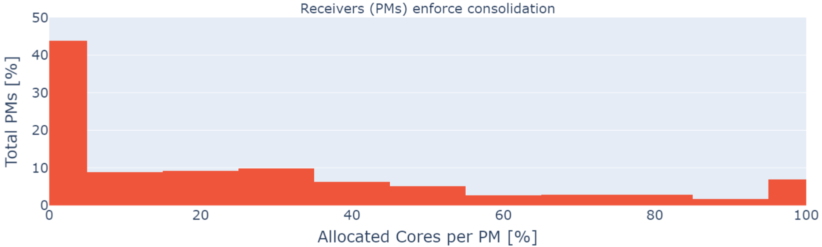


Table of Contents

6 Conclusions

UNIVERSITÀ DEGLI STUDI FIRENZE

Conclusions

- evaluated VM-to-PM matching algorithms for MEC resource allocation
- Greedy methods are fast and promote good consolidation
- Gale-Shapley and Auction-based algorithms provide better energy efficiency and adaptability via dynamic preferences and pricing
- in Gale-Shapley, consolidation placement (in proposer vs. in receiver) notably influences final allocations

UNIVERSITÀ DEGLI STUDI FIRENZE

References

[1] Long Zhang et al. "Virtual Resource Allocation for Mobile Edge Computing: A Hypergraph Matching Approach". In: *2019 IEEE Global Communications Conference (GLOBECOM)*. 2019, pp. 1–6. DOI: 10.1109/GLOBECOM38437.2019.9013384.

[2] David F. Manlove. "Hospitals/Residents Problem". In: *Encyclopedia of Algorithms*. Ed. by Ming-Yang Kao. Boston, MA: Springer US, 2008, pp. 390–394. ISBN: 978-0-387-30162-4. DOI: 10.1007/978-0-387-30162-4_180. URL: https://doi.org/10.1007/978-0-387-30162-4_180.

[3] G. Piqué. *Pikerozzo / matching-service-placement. https://github.com/Pikerozzo/matching-service-placement.* (19.05.2025).