UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

An analysis of Genetic Algorithms to support the management of edge computing infrastructures

Authors

Claudia Canali - *claudia.canali@unimore.it*
Riccardo Lancellotti - *riccardo.lancellotti@unimore.it*
Riccardo Mescoli - *riccardo.mescoli@unimore.it*

NCA 2024 - 25/10/2024 - Bertinoro, FC, Italy

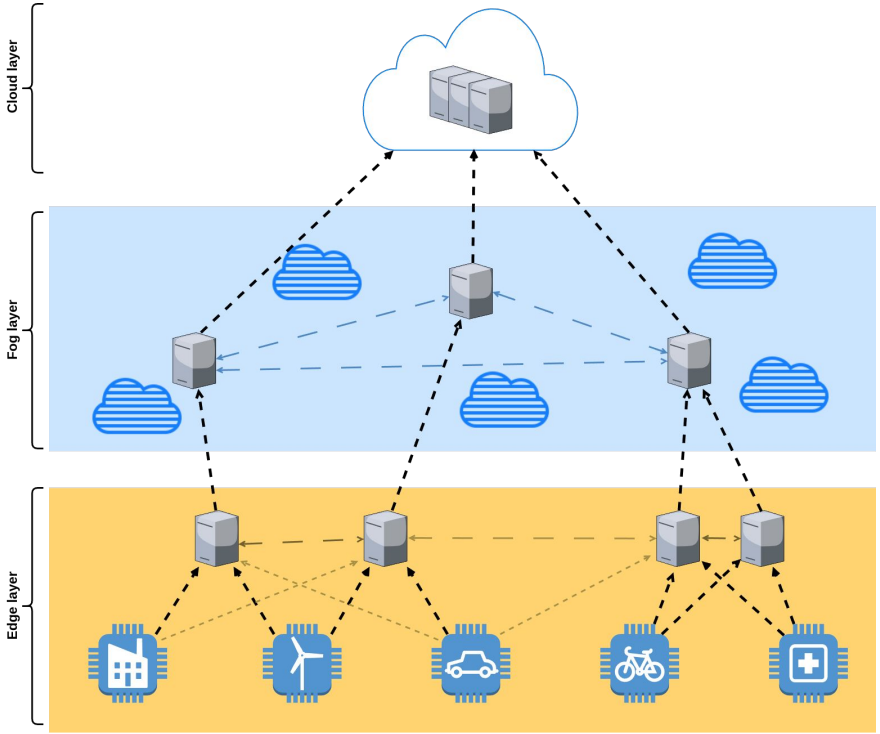
Table of contents

1. Introduction

Edge Computing Overview

- Computation closer to sensors and users
- Important for modern mobile applications and IoT systems

- Computationally limited nodes
- Non-negligible network delays

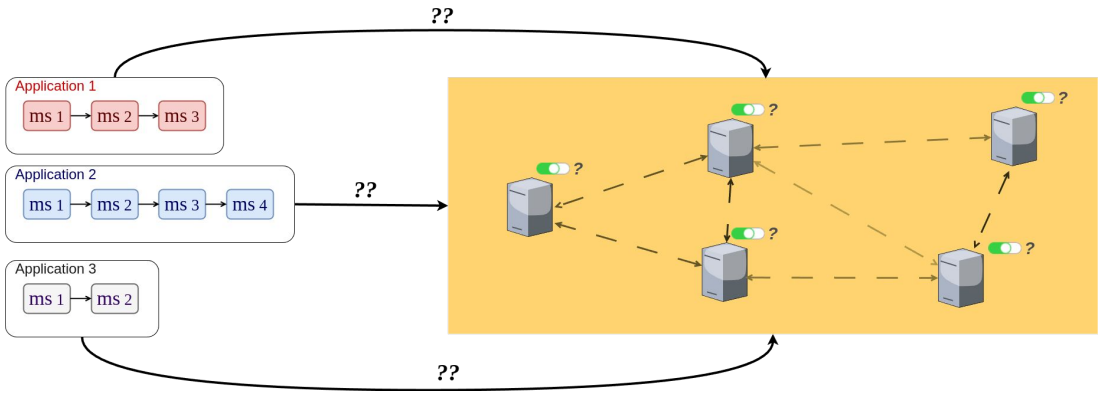


Orchestration in Edge infrastructure

- Applications designed as a chain of microservices to allocate on edge nodes
- Multi-objective optimization problem:
  1. Minimize power consumption (Energy)
  2. Minimize the average application response time (Performance)

- In the function objective 1 (energy) has priority over 2 (performance):

$$Obj = Energy + \epsilon \cdot Performance$$



Problem Overview

- The power consumption as a linear function depending solely utilization:

$$P_e = y_e P_e^0 + (P_e^M - P_e^0)\lambda_e S_e$$

- Performance modeled using queue theory as a M/G/1 system:
  - Arrival process: Poisson distribution
  - Queue waiting time: Pollaczek-Khinchin formula
  - Response time is a combination of network delays, queue wait times and execution time
- Constraints enforce no node overload, Service Level Agreements and service allocation to a single node.
- Problem is non linear, non convex, NP-hard

$$\min \mathrm{En} = \sum_{e \in \mathcal{E}} P_e$$

$$\min \mathrm{Perf} = \sum_{a \in \mathcal{A}} \frac{\lambda_a}{\Lambda} R_a$$

subject to:

$$\sum_{e \in \mathcal{E}} x_{m,e} = 1 \quad \forall m \in \mathcal{M},$$

$$\lambda_e S_e \leq y_e(1-\epsilon) \quad \forall e \in \mathcal{E},$$

$$R_a \leq T_a^{SLA} \quad \forall a \in \mathcal{A},$$

$$x_{m,e}, y_e \in \{0,1\}, \quad \forall m \in \mathcal{M}, e \in \mathcal{E},$$

2. Genetic Algorithm Solver

Genetic Algorithm

- Meta-heuristic optimization
- Solutions encoded as a sequence of symbols called chromosome
- Defined by its evolutionary strategy, selection, crossover and mutation operators

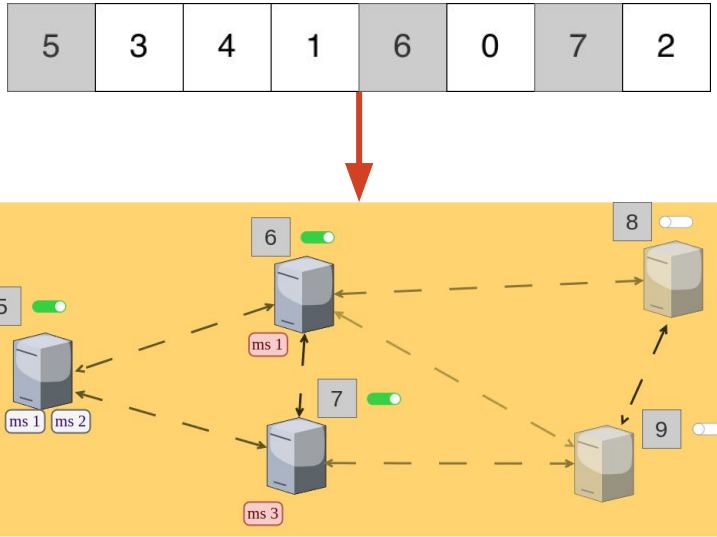
- Fitness function defined as the objective function with penalties for constraint violations
- Our implementation adopts Simple Strategy as an evolutionary strategy and Tournament selection as a selection operator, both well known in literature.

Chromosome solution encoding

- Sequence of genes that encode a solution
- Each gene represents:
  - A microservice: $ID \in [0, M-1]$

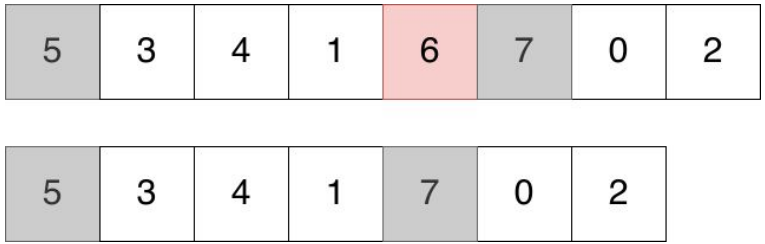
    M is the total amount of microservices
  - An edge node: $ID \in [M, M+E-1]$

    E is the total amount of edge nodes
- The sequence has a positional encoding
  - Every microservice is placed on the closest edge gene from the left in the sequence
- The first gene must always be an edge node (this must be enforced during crossover and mutation)



Normalization operator

- Normalizing a chromosome is the process of removal of genes representing empty edge nodes
- The benefits are the removal of unnecessary information and the incentive to converge towards energy efficient solutions.
- As a consequence, the genetic operators must be modified in order to support working with chromosomes of different length.



Mutation Operator

Modified Shuffle mutation:

- Takes a random gene in the sequence and swaps its value with one of the following genes.
- In order to preserve the validity of the encoded solution, the first gene can be swapped only with other edge node genes.

Modified Shuffle Mutation

$C = \{c_i\}$ *a chromosome*

$\zeta_M = [0,\ M - 1]$ *where M is the number of microservices*

$\zeta_\xi = [M,\ M + E - 1]$ *where E is the number of edge nodes*

Pick a random index $i < |C| - 1$

If i ***is first gene (*** $i = 0$***) :***

Pick a random intex $j : i < j < |C|,\ c_j \in \zeta_\xi$

Else:

Pick a random intex $j : i < j < |C|$

Swap the values of c_i *and* c_j

Normalize the chromosome C

Crossover Operator

- **Modified ordered crossover:**
  1. Extract from the parents one sub-sequence each, delimited by two random indexes
  2. For every subsequence initialize a child sequence inheriting the sub-sequence and the first gene
  3. Fill empty positions in the sequence with new genes from the other parent
  4. In case there are remaining empty spots: repeat step 2 with the parent of the inherited sub-sequence

Modified Ordered Crossover

C_{p1}, C_{p2} *are the parent chromosomes*

$\zeta_M = [0,\ M - 1]$ *where M is the number of microservices*

$\zeta_\xi = [M,\ M + E - 1]$ *where E is the number of edge nodes*

1. *Initialize two child sequences* C_{ch1}, C_{ch2} *of length* $|C_{p1}|$ and $|C_{p2}|$
2. Generate two random indexes l_{st}, l_{end} such that:
$$1 \leq l_{st} < l_{end} < min(\{|C_{p1}|, |C_{p2}|\})$$
3. $c_0^{ch1} \leftarrow c_0^{p1}$, $c_0^{ch2} \leftarrow c_0^{p2}$
4. *Fill empty genes in* C_{ch1} (C_{ch2} resp.) with new genes from C_{p2} (C_{p1} resp.)
5. **If child has no more empty genes but parent still has new genes:**
   Append new genes
6. **Else if child has remaining empty genes:**
   Repeat steps 4. and 5. with the other parent sequence

Normalize C_{ch1}, C_{ch2}

Parent sequences w/ highlighted sub-sequences

6	4	5	2	8	1	7	3
8	5	2	6	3	1	7	4

Resulting child-sequences

6	3	5	2	8	1	7	4
8	4	2	6	3	1	5	7

3. Experimentation

Experimental setup

- Each scenario tested has undergone 30 runs over networks with randomly generated delays
- The Genetic Algorithm (GA) was run with a population of 600 and 600 max. generations

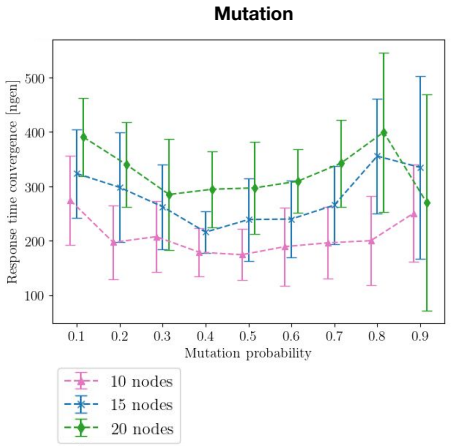
- The default scenario used is set with 15 edge nodes, 9 applications made of 10 microservices and system utilization (ρ) of 0.6.

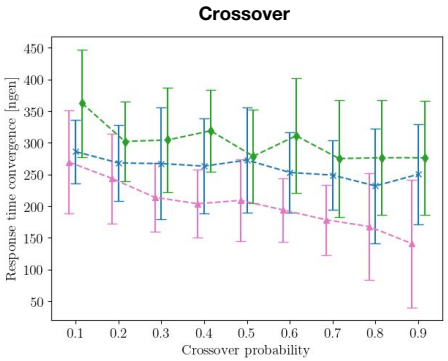
- The main metrics analyzed have been the two components of the objective function (power consumption and response time) along with the number of generations necessary for the secondary objective to converge.

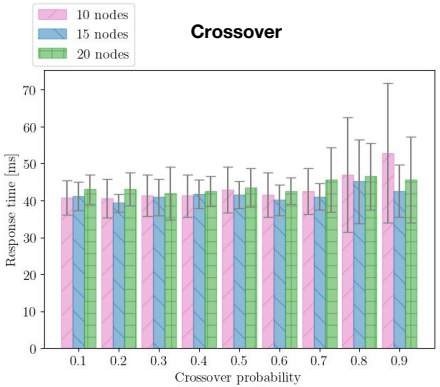
- The aim of the experimentation has been to find a configuration of GA parameters suitable for a wide range of problem sizes (defined by the number of edge nodes given a fixed ρ) and test its robustness and scalability.

GA Parameter Tuning

- The parameters tested were the mutation and crossover probabilities
- Tested over 3 problem sizes (10, 15, 20 edge nodes)
- Analyzed the convergence and response times* in relation to the probability values
- The results lead to the choice of values 0.4 for mutation and 0.5 for crossover



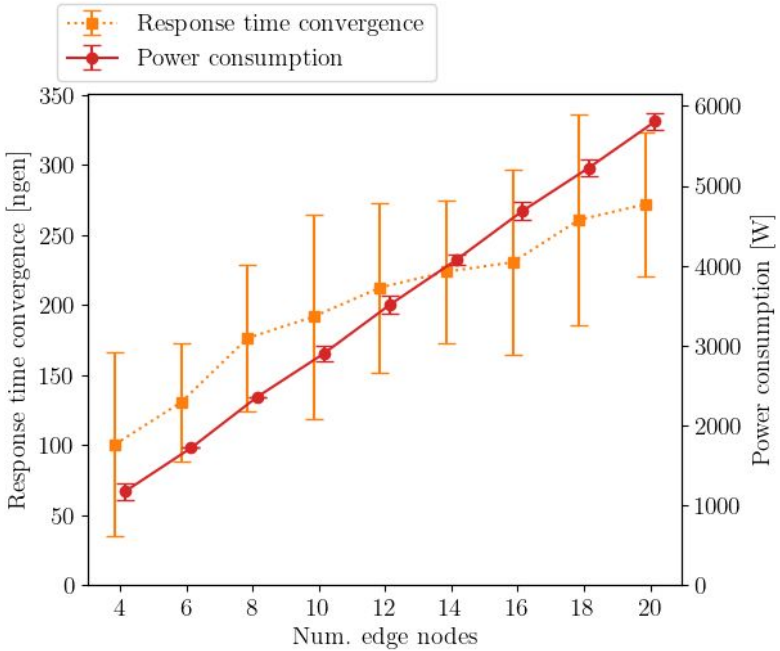




*Relevant for the choice of crossover probability

Scalability

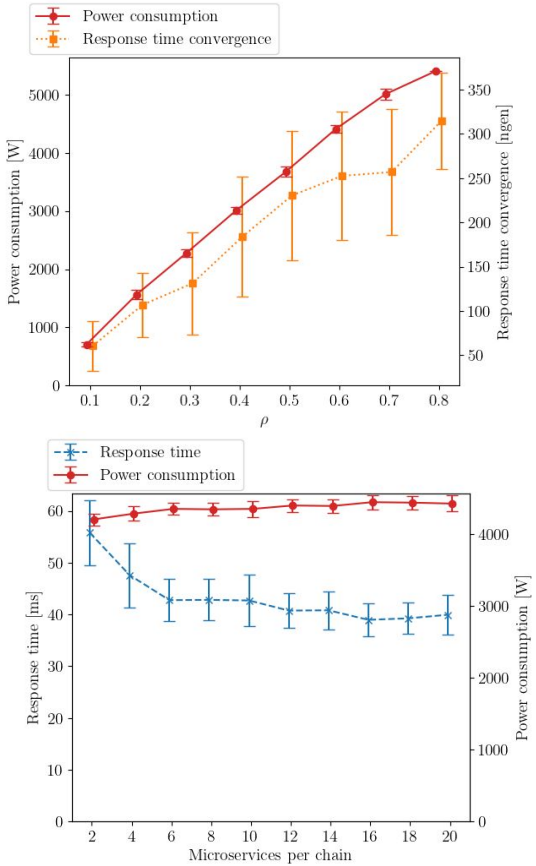
- To test the system for scalability we observe power consumption (red line) and convergence (orange line) in relation to a wide range of problem sizes (4-20).
- Power consumption (red) increases linearly (due to fixed ρ) and proves to converge reliably to the same value (error bar - stdev).
- Although the convergence time (orange) grows sublinearly and it still manages to identify the best solution in a few hundred generations



Robustness

- **System utilization - ρ (top):**
  Analyzing power consumption (red) and response time convergence (orange) we get observations similar to what we obtained testing for scalability
  - Power consumption increases due to the system utilization but still manages to converge consistently to equivalent solutions
  - Convergence time gets longer as the area of unfeasible solutions increases, but still occurs in around 300 generations at most

- **Num. microservices per chain (bottom):**
  - With less, larger microservices it results harder to load balance while, as the number increases, the system rapidly achieves better response times (blue line)
  - This also displays the system's ability to prioritize power as, given ρ is fixed, power consumption (red) stays fairly constant



4. Future work

Future Work

- Test other classes of genetic algorithms
- Define the pareto front for the energy/performance trade-off
- Explore a wider set of problem characteristics (e.x: impact of network delays on algorithm stability)