Placement of IoT Microservices in Fog Computing Systems: A Comparison of Heuristics

Bilel Arfaoui

224618@studenti.unimore.it

University of Modena and Reggio Emilia

2024

Introduction

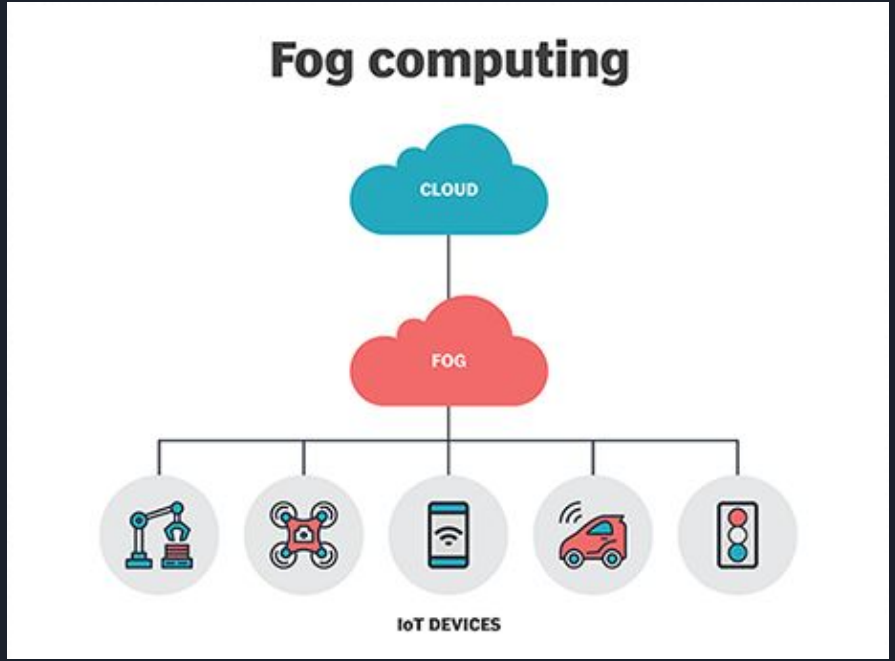
This presentation explores IoT microservice placement within a Fog network, that involve geographically distributed nodes and sensors, focusing on Fog computing solution. We'll also discuss related challenges and placement strategies within SLA constraints.

Fog computing

Fog computing brings computational resources closer to end users, improving response times, and reduce data sent upstream.

Hower **cloud-based solutions** often fall short in Fog environments due to several key differences:

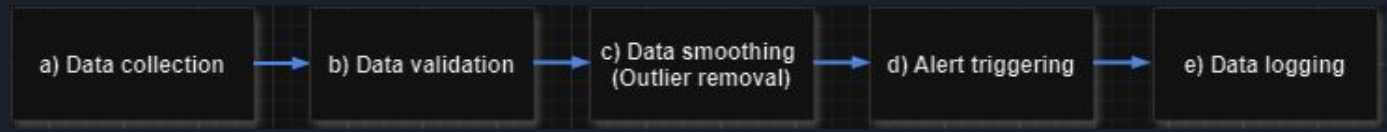
- High heterogeneity of nodes
- Limited computational resources
- Significant delays between Fog nodes
- Centralized nature of cloud solutions
- Network congestion



Microservices chains

In the microservice paradigm, applications are broken down into small, independent services, and when two or more of these services are interconnected, they form **microservice chains**.

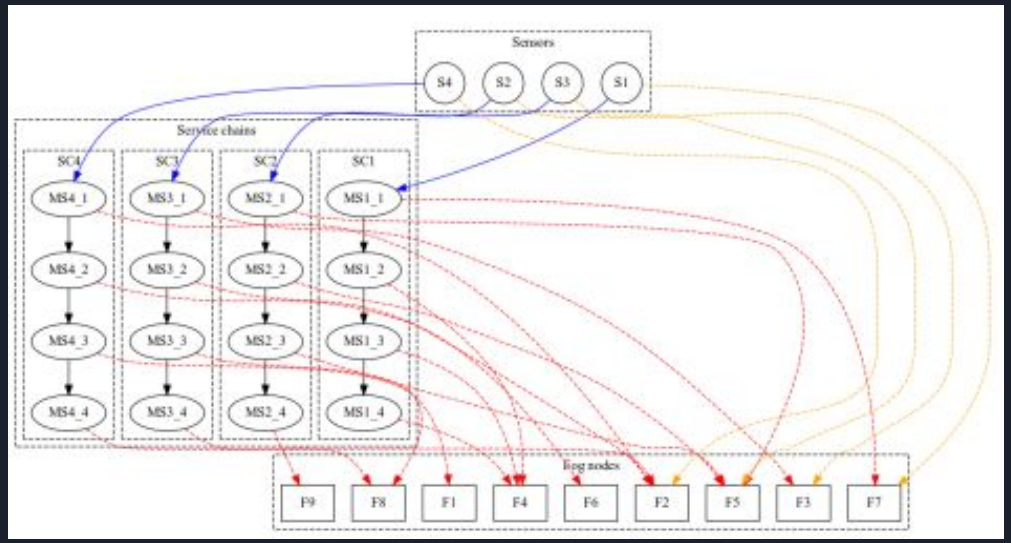
Example: Microservice chain used for monitoring and logging sensor data in a industrial plant



The microservices of a chain may be allocated over one or multiple fog nodes, and a host may have microservices of different chains

BACKGROUND AND PROBLEM DEFINITION

Placement problem



Inputs to the problem

- Iot applications modeled in |C| microservice chains with related SLAs
- Each chain **“c”** is composed by **|Lc|** microservice
- **|F|** Fog nodes

Objectives

- Minimize the response time
- Comply to the SLAs

Background

Assumptions:

- Stable Request rate
- A microservice is allocated to **1 and only Fog node**
- Message length is small
- The delay is the sum of latency and transmission time
- The SLAs are based on the average service time of the microservices.

Note: some assumption are made for the sake of simplicity

PROBLEM FORMALIZATION

Objective function:

$$\min obj(x) = \sum_{c \in C} \omega c * Rc$$

- Rc is the response time of a single chain.
- Wc is the weight given to proportional to the activation frequency.

Constraints:

- each microservice is allocated to 1 and only fog node
- each fog node must not be overloaded
- each microservice must respect the SLA

CHAIN RESPONSE TIME

The response time **Rc** of the chain is the sum of the following contributions:

1) The sum of waiting time for each node

$$\sum_{m \in C} \sum_{f \in F} W_f * X_{m,f}$$

2) Sum of average service times

$$\sum_{m \in C} S_m$$

3) Network delays due to data transfer among microservices in the chain

$$\sum_{m1,m2 \in C} \sum_{f1,f2 \in F} o_{m1,m2} * \delta_{f1,f2} * X_{m1,f1} * X_{m2,f2}$$

How will we tackle the placement problem?

Due to its complexity caused by the large number of potential microservice/resource combinations, the placement problem is **NP-hard**, for this reason we will focus on applying the following heuristics:

- **Modified Best Fit Decreasing (MBFD)**
- **Genetic Algorithm (GA)**
- **Variable Neighborhood Search (VNS)**

Afterwards we will compare their performance to evaluate their efficiency in achieving our objectives.

Modified Best Fit

This heuristic has been used for the placement of VMs in physical hosts, in this case we adapted it for the placement of microservices over the Fog Nodes.

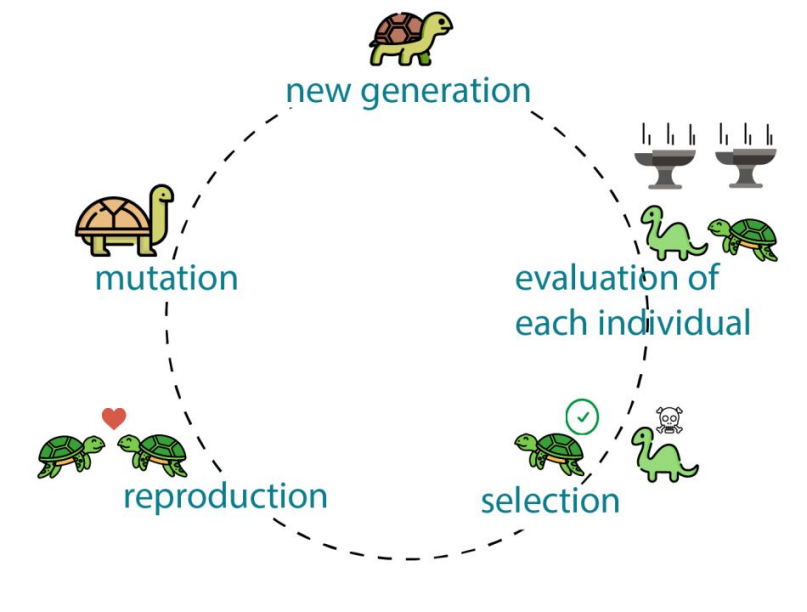
Iteration:

- Iterate over **|M|** microservices in descending order of mean service time.
- Initiate the placement process for each microservice in order.

Placement process:

- Iterate over the **|F|** Fog nodes
- Assign the microservice to the node that best optimizes the objective function while meeting constraints and SLA requirements.

Genetic Algorithm

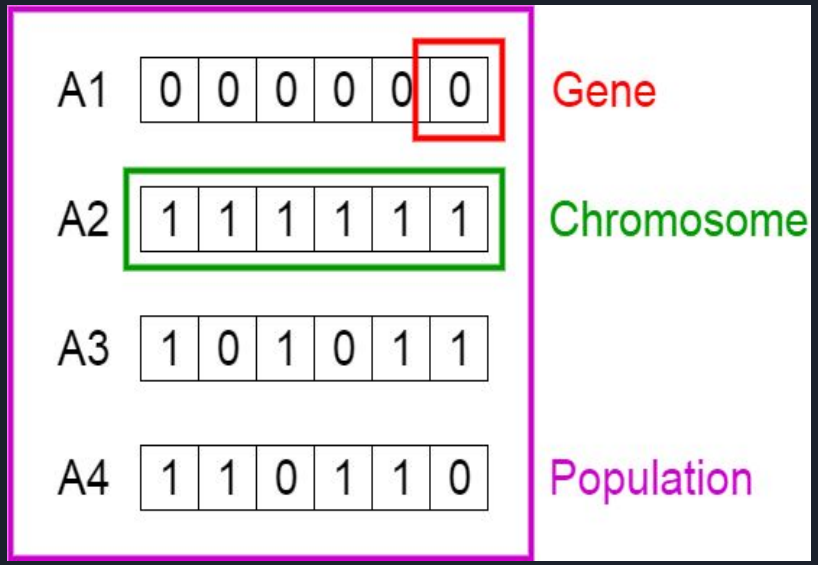


The **Genetic Algorithm** is a Biology inspired Heuristic where the solution space is a population of individuals, and iterates over the following phases:

- **Selection:** The fittest are select to pass their genes.
- **Mutation:** Random changes to individual genes.
- **Crossover:** To create new offspring with mixed traits.

Each individual has a fitness score assigned by the fitness function.

Genetic Algorithm - Setup



As said before the solution space is composed by individuals, they are known as **chromosomes**, each consisting of individual elements called **genes**.

We model the solution space, where each chromosome has **|M|** genes with values ranging from 1 to **|F|**.

The **fitness function** in our study will be the objective function.

Genetic Algorithm - Setup

The algorithm runs for 600 generations over 60 individuals with the following strategy:

- Tournament selection
- Random mutation
- Uniform crossover
- Objective function acts as Fitness Function with added penalties.

Both the mutation and crossover happen with a probability of **0.8%**.

VNS - Variable Neighbourhood Search

The VNS heuristic focuses on finding the optimal solution exploring the solution space with local search and diversification strategies to escape local optimas.

The algorithm start by defining an **initial solution** continues as the following:

- **Perturbation**: Drastic change of the solution to escape local optima and explore new regions.
- **Local Search:** Explores the neighborhood for improvements.
- **Iterate:** Until a stopping criterion is met

VNS - Solution space

The **Solution Space** is expressed as the following recursive dictionary:

Microservices = {

C1: { M1: Fog1, M2: Fog2, M3: Fog3},

C2: { M4: Fog5, M5: Fog7, M6: Fog11},

C3: { M7: Fog6, M8: Fog8, M9: Fog1},

…. }

The algorithm starts with all microservices placed to one node, and then after a perturbation the local search starts.

VNS - Local Search

The phase is composed by the following structures:

- **N1***: Swapping microservices between two Fog Nodes
- **N2***: Random selection of a random busy Fog Node, and then migration of its allocated microservice to the least loaded Fog Node

The **local search** phase select which microservice to move based on:

- **Feasibility**: SLA compliance etc..
- **Distance**: Delay, number of hops from the data source and/or the traffic volume in the network.

VNS - Perturbation

The **perturbation** phase is composed by the following structures:

- **L1**: Perform every possible swap between the Fog nodes
- **L2**: Perform every possible allocation

Stopping criterion: There is no improvement during the local search phase after the perturbation

Performance evaluation

Analysis setup

Performance Assessment

- **Efficiency**: Time taken to achieve a solution.
- **Solution Quality**: Evaluating the effectiveness of the solution.

Metrics

- Problem Size: **|Lc|**, **|F|** and **|C|**
- Jain Index: Measures load balancing capability of the algorithm [0,1].
- Number of Hops normalized to the chain Length
- Overall Load (ρ)

Note: During the analysis, we'll adjust specific parameters as needed to better understand performance.

Scalability - Setup

In our initial experiment, we will evaluate the algorithms' capacity to manage load balancing as the number of nodes increases.

To maintain an even distribution of load across the nodes, we will proportionally increase the number of chains as the node count grows.

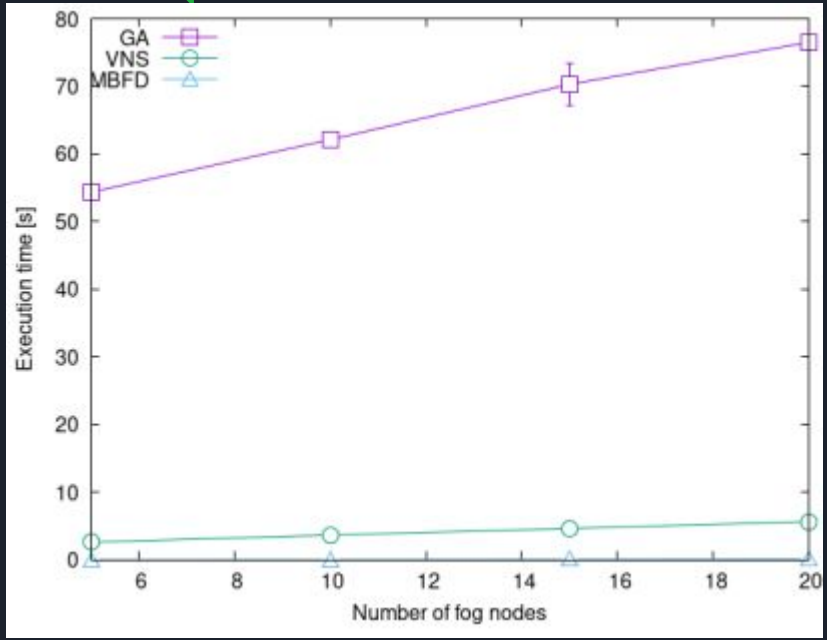
Fixed parameters

$|Lc|=5$

$\rho = 0.6$

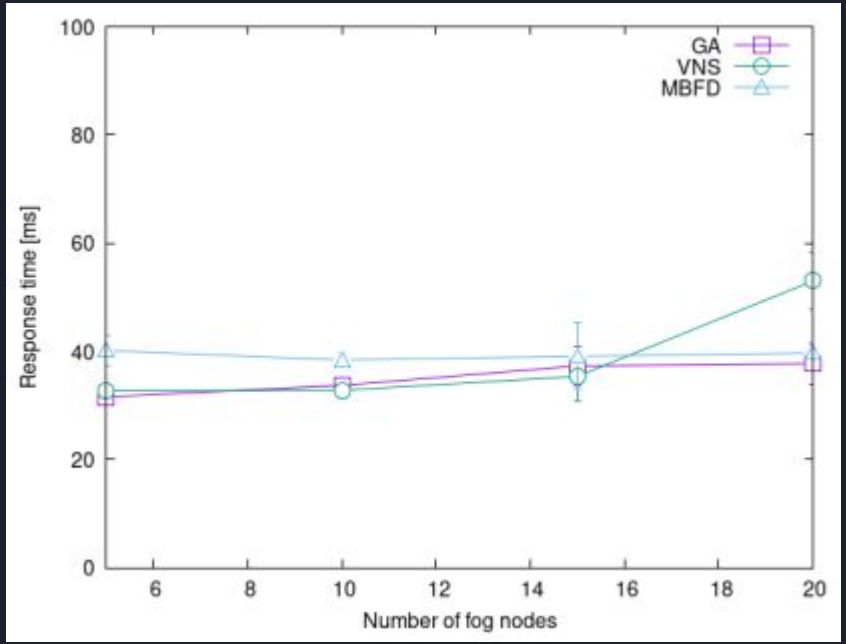
$Tsla = 10 * \sum_{m \in M} S_m$

SCALABILITY - Execution time



- **GA's is due to** the size of the population and high number of generations.
- **MDBF** is almost negligible thanks to its greedy approach.
- **VNS** keeps the size of the neighbourhood small, so their exploration is fast
- The increase of the execution time is linear

SCALABILITY - Response time



- **MBFD** provides a stable performance
- **GA** and **VNS** experience degradation

System load - Setup

As for this analysis we will analyze the impact of the global system load over algorithms' ability of load balancing, optimization of the objective function and the ability of keeping the hops low between subsequent microservices.

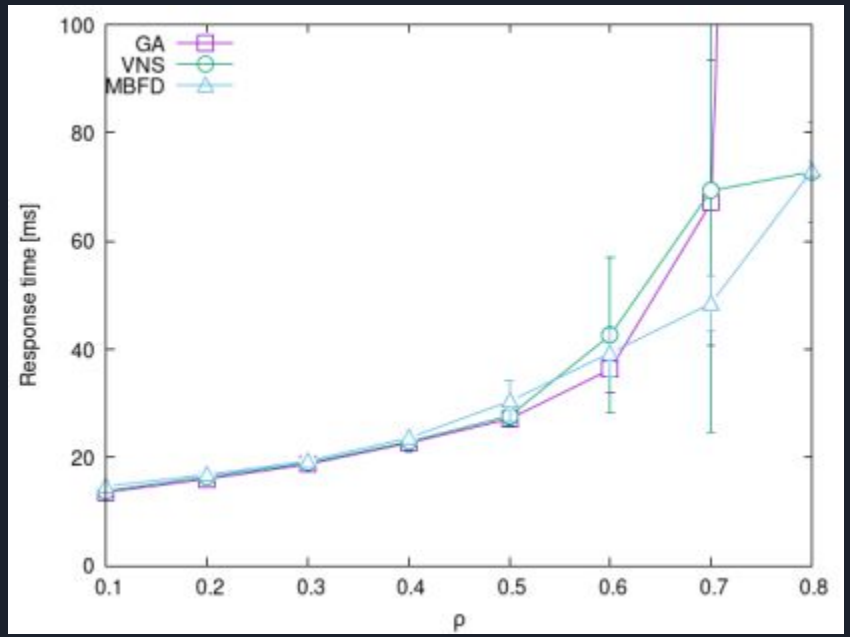
Fixed parameters

|Lc|=5

|F| = 10 fog nodes

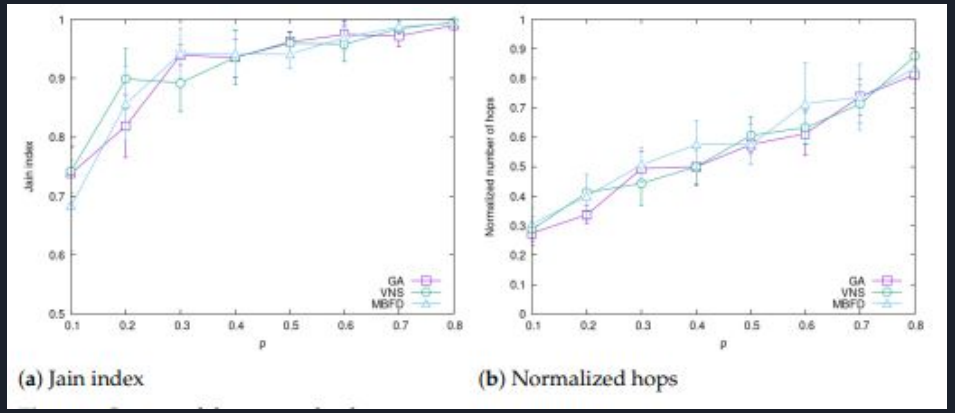
$Tsla = 10 * \sum_{m \in M} S_m$

System load - Response time



- On low values they are all comparable.
- As the load increases the variance and the response time increase with it.
- MBFD performs well under heavy work loads.

System Load - Jain Index and Normalized Hops



(a) Jain index (b) Normalized hops

- **MBFD** has a consistently good performance due to it being designed to to guarantee good load balancing
- The **number of hops** increases because the placement of the microservices becomes harder to place near each other without risking overload
- Jain index in the beginning is lower and this may be caused by underutilized nodes

Service chain Length - Setup

For the final test, we will assess the impact of increasing the length of microservice chains while keeping the global load constant.

This approach will distribute the load more broadly, allowing us to evaluate how the algorithms perform under these conditions.

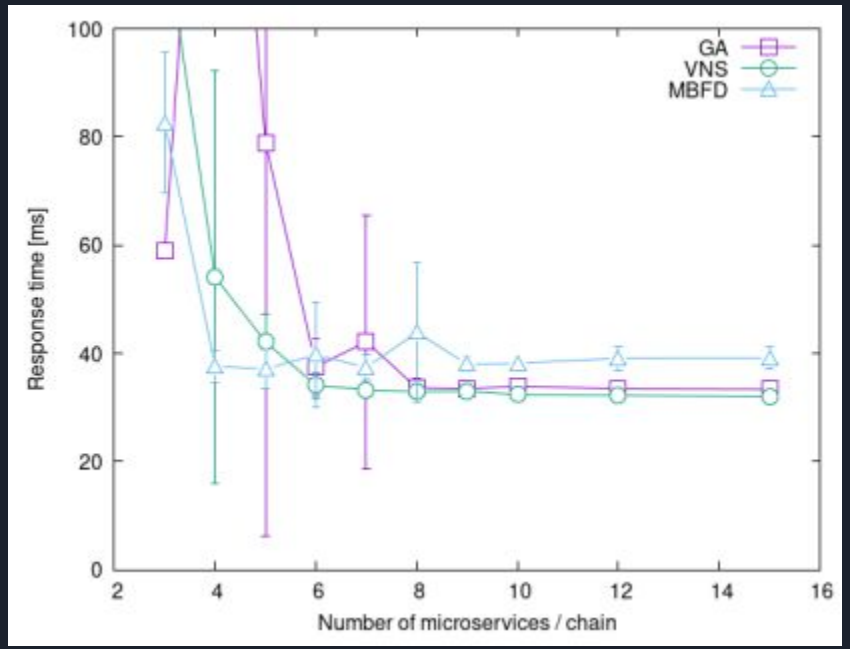
Fixed parameters

|F| = 10 fog nodes

ρ = 0.6

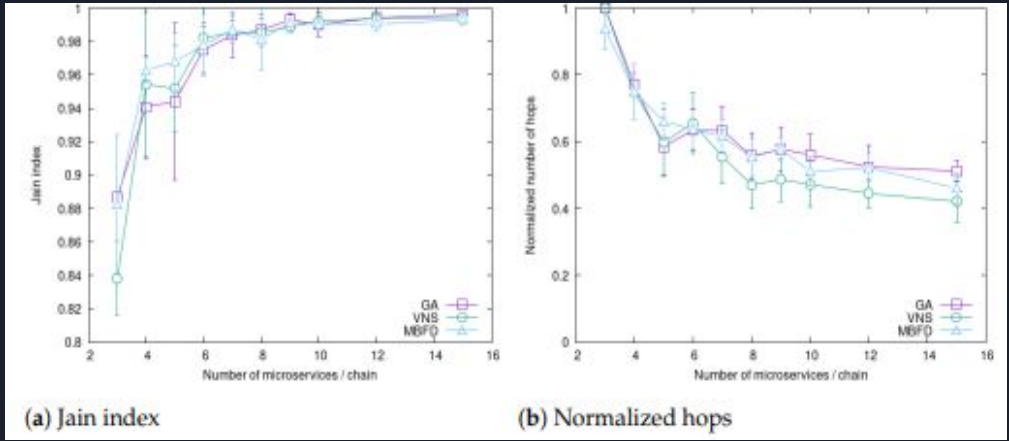
$Tsla = 10 * \sum_{m \in M} S_m$

Service chain length - Response time



- As the length increases the variance and quality improve
- As the chain length increase the load is more spread out, stabilizing the performance and improving the load balancing

Service chain length - Jain Index and Normalized Hops



(a) Jain index (b) Normalized hops

- **MBFD** provides good balancing even in short chains, but does reduce hops very well
- **GA** does not perform well in reducing the hops
- **VNS** shines here with a high Jain score and low hops

Conclusions

Pros VS **Cons**

Modified Best Fit
Works really well under light/heavy loads, focusing on load balancing, having the best execution time.

Genetic algorithm
Good response time when scaling up the number of nodes and/or the length of the service chains.

VNS
Good execution time, works very well over chain with light microservices, providing good load balancing and optimized hops.

Modified Best Fit
Does not take delays into account, causing lower quality solutions.

Genetic algorithm
Degradation and high variance at presence of heavy loads/microservices. High execution time.

VNS
Degradation and high variance at presence of heavy loads/microservices. Also has the worst response time when scaling up.

Thank you for your attention!