# Optimisation problem on Fog Node

*A problem description*

Rachel Roux, Octave Cinquanta

## 1 Introduction

Implementation of the paper Multi-objective optimization for task offloading based on network calculus in fog environments [1].

- Due to the exponential improvement in Big Data, including the Internet of Things, autonomous vehicles or smart cities, the number of real-time data (streams) that need to be processed quickly has exploded. Cloud computing provides many resources for massive calculations. However, centralized cloud computing is overloaded with streaming data, resulting in increased energy consumption and delays.
- For this reason, fog computing is now being used, with the architecture allowing you to determine where the computation is to be performed: on the local machine, on the fog layer or on the cloud layer. The fog layer comprises routers, switches and micro-servers that are closer to the local machine than to the cloud server, and which can perform the computation. The fog layer is made up of fog nodes (FNs). Each FN has different parameters to take into account.
- Our aim is to develop a program that associates FNs with the tasks to be performed. We assume that the total number of FNs can perform the task divided into M subtasks by the fog computation. We want to find a FN allocation that minimizes total energy consumption and delay.

**Affiliation** [1] CY Tech
**Correspondence** rouxseveri@cy-tech.fr
**Version** April 8, 2024

## 2 Problem formulation

We want to find the best vector X that minimise the power consumption, the delay and the load.

$$\begin{cases} \min\limits_{X_i} & \sum\limits_{i=1}^{N} X_i (W_1 T_{1,i} + W_2 T_{2,i} + W_3 T_{3,i}) \\ s.t. & \sum\limits_{i=1}^{N} Xi - M = 0 \\ & W_j \in [0,1], \forall j \in \{1,2,3\} \\ & T_{j,i} \in [0,1], \forall j \in \{1,2,3\}, \forall i \in [[1,N]] \\ & X_i \in \{0,1\}, \forall i \in [[1,N]] \end{cases}$$

with :

$$T_{1,i} = \frac{P_i - P^{min}}{P^{max} - P^{min}} \qquad T_{2,i} = \frac{B_i}{B_{i,S}} \qquad T_{3,i} = \frac{D_i - D^{min}}{D^{max} - D^{min}}$$

We assume that $P_i, P^{min}, P^{max}, B_i, B_{i,S}, D_i, D^{min}, D^{max}$ are internal parameters linked to the $i^{th}$ FN itself that we can get easily, the $X_i$ are our binary decision variables. Some objective functions want to minimize power consumption ($T_{1,i}$) and delay ($T_{3,i}$). $T_{2,i}$ wants to minimize the buffer size $B_i$ (waiting data to be processed) from the total remaining storage space $B_{i,S}$. If the buffer size is bigger than the total remaining storage, then the FN cannot be allocated. Which means it is a hard constraint.

I will choose to change this minimization function to a hard constraint. It would also give me a best 2D representation of pareto front and I remove the aggregation.

Finally, we have:

$$\begin{cases} \min\limits_{X_i} & \sum\limits_{i=1}^{N} X_i T_{1,i} \\ \min\limits_{X_i} & \sum\limits_{i=1}^{N} X_i T_{3,i} \\ s.t. & \sum\limits_{i=1}^{N} Xi - M = 0 \\ & \sum\limits_{i=1}^{N} T_{2,i} - 1 < 0 \\ & W_j \in [0,1], \forall j \in \{1,3\} \\ & T_{j,i} \in [0,1], \forall j \in \{1,2,3\}, \forall i \in [[1,N]] \\ & X_i \in \{0,1\}, \forall i \in [[1,N]] \end{cases}$$

In this formulation, we assume that each Task is the same, which means, whatever the task chosen on the FN, the $P_i$ and $D_i$ are the same. But more precisely, it is not the same and a FN can process different task while the buffer size is still available. Hence, we change the formulation for this one :

$$\begin{cases} \min\limits_{X_{i,j}} & \sum\limits_{i=1}^{N} X_{i,j} P_{i,j} \\ \min\limits_{X_{i,j}} & \sum\limits_{i=1}^{N} X_{i,j} D_{i,j} \\ s.t. & \sum\limits_{i=1}^{N} X_{i,j} - M = 0 \\ & \sum\limits_{i=1}^{N} \frac{B_{i,j}}{B_{i,j,S}} - 1 < 0 \\ & \sum\limits_{i=1}^{N} X_{i,j} = 1 \quad \forall j \\ & X_i \in \{0,1\}, \forall i \in [[1,N]] \end{cases}$$

Where $X_{i,j}$ is the allocation variable from task i to FN j with $P_{i,j}$ and $D_{i,j}$ the Power consumption and Delay linked. The first constraint still permits to allocate every task to FN and the second permits to be sure about the availability of the FN. The new constraint permits to be sure that each task is allocated to exactly one FN.

## 3 Dataset

The first idea we had was to recreate the dataset. We assume that the constraint of the buffer size is respected. $P_i = P_i * frac\tau_i f_i$ with $f_i$ the CPU frequency, $\tau_i$ the

number of CPU cycles to process 1 bit-data and $P_i*$ the power consumption for one cycle. Each task is determined by l bit-data, so the power $P_{i,j} = lP_i$. For the Delay, the article uses the available service curve where the service curve represents the disposition of the FN to process the task and the arrival curve representing the task characteristics. This solution needs to process some network calculus. An example of matrix of allocation was depicted :

Table 1. Node-task table.

| FN/ST | 1 | 2 | 3 |
|---|---|---|---|
| A | – 0.3/1 | – 0.1/2 | – 0.4/2 |
| B | – 0.2/2 | – 0.1/3 | – 0.3/2 |
| C | – 0.1/1 | – 0.1/3 | – 0.5/2 |

This is why the task offloading dataset where used because similarly we could see two matrix of allocation $P_{i,j}$ and $D_{i,j}$.

## 4 Algorithm

We can choose our algorithm from this list : Firefly Algorithm (FA), Harmony Search Algorithm (HS), Bat Algorithm, Cuckoo Search (CS), Dragonfly Algorithm, ACO, Bacterial Foraging Algorithm (BFOA), Levy flight, hybrid meta heuristics + machine learning/deep learning.
We choose Levy Flight and Monarch Butterfly.

### 4.1 Levy Flight

This algorithm is inspired by the erratic behavior of animals as they search for food, taking small steps and then large steps less frequently to modify the total environment. By mimicking this behavior, our algorithm seeks to avoid getting stuck in local minimums.

This algorithm is not implemented for discrete optimization problems. To match

the Levy Flight algorithm to the specific problem, we defined the process steps as follows: The main feature of this method is the number of times the solution

**Algorithm 1** Levy Flight

```
n: int
Stop_Criteria: int
Generate a list L of N variables with random values from available FN
List_Solution ← L
while len(List_Solution) < Stop_Criteria do
    p ← rand.normal(μ, σ)
    if p < nσ then
        Operate few random modification into L
    else
        Operate lot random modification into L
    end if
    if !isDominated(L, List_Solution) then
        List_Solution ← L
    end if
    if Dominate(L, List_Solution) then
        List_Solution.remove(solution)
    end if
end while
```

is modified according to the probability of a Gaussian, which is inspired by the probability of a bird making large jumps. The further the random number is from the established mean, the greater the probability of randomizing the entire existing solution.

#### 4.1.1 Parameters :

Normal distribution with mean 0 and standard deviation $\sigma$ from 0.25 to 0.75
Chance to shuffle the elements of the solution : Number p generated on the Gaussian :

- If |p| >= 3*$\sigma$ (1% chance) we change the whole solution.
- If |p| >= 2*$\sigma$ (5% chance) we change almost 75% of the solution.
- If |p| >= $\sigma$ (50% chance) we change 50% of the solution.

- Else we change 25% of the solution.

### 4.2 MONARCH

This algorithm is inspired by the migratory behavior of Monarch butterflies when they have to orient themselves in new environments, by following certain individuals who have a better probability of orienting themselves. By imitating this behavior, our algorithm seeks to retain a long-term memory of which behaviors to adopt.

The Monarch algorithm is clearly used for real-variable optimization problems. To match the Monarch algorithm to the specific problem, we defined the process steps as follows (see page 8)

The main feature of this method is the probability of copying elements coming directly from one of the best solutions (from the pareto front or from the rank closest to the pareto front), guaranteeing long-term memory of the movements to be performed. This probability follows a Levy distribution.

#### 4.2.1 Parameters :

Uniform distribution with mean 0 and standard deviation $\sigma$ from 0.25 to 1
Chance to shuffle the elements of the solution : Number p generated on the Gaussian :

- Total population at each pass: 20
- Sub-population NP1 = NP2 = 10
- Parameter p: threshold below which a random number of each monarch genome must be generated to copy a genome from one of the best solutions
- Number of algorithm iterations: varies between 5 (minimum for Front) and 25 (too much convergence into local minimum)

## 5 Result discussion

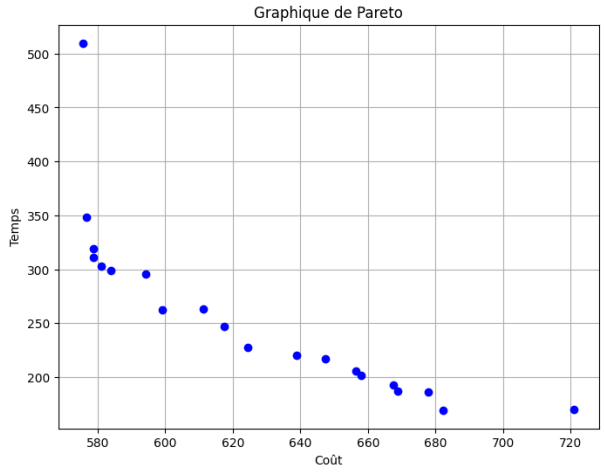
For 40 tasks spread over 13 NF:

| Comparison | Monarch | Levy Flight |
|---|---|---|
| Average time | 3.5 | 2.5 |
| Finding Front Pareto | depending on the coefficient: Front pareto or trapped solution intot local minimum | ordered solution in Pareto front form |
| Variation factors | Probability of permutation with one of the best Monarch | Gaussian flattening |

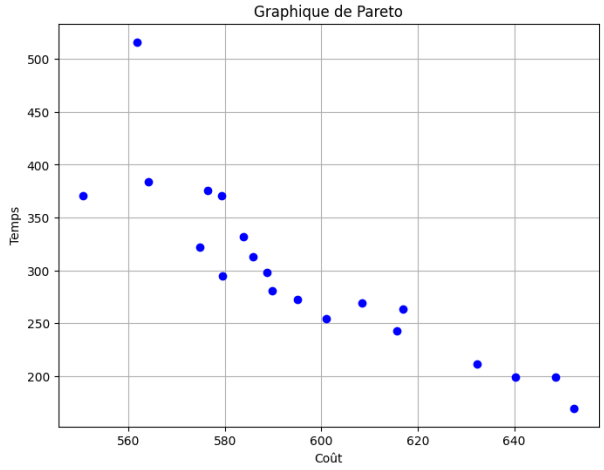
**Algorithm 2** Monarch

n: int
$stop \leftarrow 0$
Stop_Criteria: int
Generate a list of Population L of N Monarch with random values from available FN on their genome.
$NP1 \leftarrow L[\frac{N}{2} :]$
$NP2 \leftarrow L[: \frac{N}{2}]$
**while** $stop < Stop\_Criteria$ **do**
    $stop \leftarrow +1$
    **for** Monarch in NP1 **do**
        $p \leftarrow rand.normal(0, 1)$
        **if** $p < n$ **then**
            Operate some permutation between Monarch into NP1
        **else**
            Operate some permutation between Monarch into NP2
        **end if**
    **end for**
    **for** Monarch in NP2 **do**
        $p \leftarrow rand.normal(0, 1)$
        **if** $p < n$ **then**
            Operate some permutation between Monarch into NP2
        **else**
            Operate some permutation between Monarch and one non-dominated Monarch in L
        **end if**
    **end for**
    Modify the list L with new of Population L from Parent Monarch and Children Monarch from rank near of Pareto Front to higher rank until N Monarch.
    $NP1 \leftarrow L[\frac{N}{2} :]$
    $NP2 \leftarrow L[: \frac{N}{2}]$
**end while**

Best Front Pareto for Levy-Flight (Gaussian (0,1)):




Best Front Pareto for Monarch (algorithm iteration : 15, probability for NP2 to copy one of the best monarch Genome : 0.25) :




## 6 Conclusion

Both algorithms have an impressive range of solutions. We could do further trials with these algorithms using different probability laws for the monarch and selecting the monarch based on crowding distance and not just using a random solution.

## References

[1] Multi-objective optimization for task offloading based on network calculus in fog environments, Qian Ren, Kui Liu, Lianming Zhang (2022).

[2] Optimal Workload Allocation in Fog-Cloud Computing Towards Balanced Delay and Power Consumption, Ruilong Deng, Rongxing Lu, Chengzhe Lai, Tom H. Luan and Hao Liang (2016).

[3] Solving 0–1 knapsack problem by a novel binary monarch butterfly optimization, Yanhong Feng, Gai-Ge Wang, Suash Deb, Mei Lu, Xiang-Jun Zhao (2015).

[4] https://knaidoo29.github.io/mistreedoc/levy_flight.html

[5] Monarch butterfly optimization, Gai-Ge Wang, Suash Deb, Zhihua Cui (2015).