In [ ]:

```python
#
#   Example 6: Finding optimum number of Edge and Cloud devices
#   (optimization model comparison)
#
#       Results and execution time of the developed optimization.py
#   model are compared to the scipy.optimize results
#
#   Author: Paulius Tervydis
#   Date: 2023-10-19
#
# ===============================================================

import numpy as np
from scipy.optimize import minimize
from qsystems import *
import time

# Initial parameters and requirements
Lambda = 1000
r_p = 0.05
P_E = 0.3
N_E = 10
T_E = 200/3600
T_E_distr = 'Determined'
B_p = 200
C_E = 0.1
N_C = 10
T_C = 100/3600
T_C_distr = 'Determined'
C_C = 0.1
C_C_pricing = "Dedicated"
W_cr = 240/3600
T_bat_cr = 8

if T_E_distr ==  'Determined':
    qs_E = 'md1'
if T_E_distr ==  'Exponential':
    qs_E = 'mm1'
if T_C_distr ==  'Determined':
    qs_C = 'md1'
if T_C_distr ==  'Exponential':
    qs_C = 'mm1'

mu_C = 1/T_C
mu_E = 1/T_E
# P_C = 1 - P_E

#--------------------------------------------------------------
# Model for scipy.optimize
# -------------------------------------------------------------
# Define the cost function and waiting time function
def cost_function(PE, NE, NC):
    Lambda_E = PE * Lambda
    Lambda_C = (1-PE) * Lambda
    lambda_C = Lambda_C/NC
    rho_C = lambda_C/mu_C
    PC = 1 - PE
    if C_C_pricing == "Dedicated":
        # C_S = N_E*C_E + N_C*C_C
        C_S_C = NC*C_C
    if C_C_pricing == "On-demand":
        # C_S = N_E*C_E + N_C*C_C*rho_C
        C_S_C = NC*C_C*rho_C
    C_S_E = NE*C_E
    # R_S = (Lambda_E + Lambda_C)*r_p
    # P_S = R_S - C_S

    R_S_E = Lambda_E*r_p
    R_S_C = Lambda_C*r_p

    P_S_E = R_S_E - C_S_E
    P_S_C = R_S_C - C_S_C

    # return 1e6-(P_S_E+P_S_C)
    return C_S_E+C_S_C  # Example cost function

def edge_battery_function(PE, NE):
    Lambda_E = PE * Lambda
    N_E_bat_cr = np.ceil((Lambda_E*T_bat_cr)/B_p)
    if NE < N_E_bat_cr:
        # return 1e6#np.inf
        return np.inf
    else:
        return 0

def waiting_time_function(PE, NE, NC):
    # W_E = 1e6#np.inf
    # W_C = 1e6#np.inf
    W_E = np.inf
    W_C = np.inf
    PC = 1 - PE
    if Lambda*PE/NE < mu_E and PE >= 0 and NE >= 0:
        E_params = msqs(ar = Lambda*PE, sn = NE, s1 = T_E, qs=qs_E)
        if E_params['w'] <= W_cr:
            W_E = E_params['w']
    if Lambda*PC/NC < mu_C and PC >= 0 and NC >= 0:
        C_params = msqs(ar = Lambda*PC, sn = NC, s1 = T_C, qs=qs_C)
        if C_params['w'] <= W_cr:
            W_C = C_params['w']

    return W_E + W_C
    # else:
        # return 1e6

# Define the combined objective function
def combined_objective(x):

    PE = x[0]
    NE, NC = np.round(x[1:]).astype(int)
    return cost_function(PE, NE, NC)+ waiting_time_function(PE, NE, NC)+ edge_battery_function(PE, NE)+(NE+NC)

# Initial guesses for PE, NE, NC
initial_guess = [P_E, 50, 50]  # Example initial guesses

# Define bounds for PE (between 0 and 1) and NE, NC (integer values)
bounds = [(P_E, P_E), (1, 1000), (1, 1000)]

# Perform the optimization
tstart = time.time()
result = minimize(combined_objective, initial_guess, bounds=bounds, method='Powell',options={'disp': True})
tfinish= time.time()
print(60*"-")
print("Results of scipy.optimize:")
t_scipy_optimize = tfinish-tstart
print("Execution time = ",t_scipy_optimize,"seconds")
optimal_PE = result.x[0]
optimal_NE, optimal_NC = np.round(result.x[1:]).astype(int)

# Calculate the corresponding cost and waiting time
optimal_cost = cost_function(optimal_PE, optimal_NE, optimal_NC)
optimal_waiting_time = waiting_time_function(optimal_PE, optimal_NE, optimal_NC)

# print(f"Optimal PE: {optimal_PE}")
print(f"Optimal NE: {optimal_NE}")
print(f"Optimal NC: {optimal_NC}")

#--------------------------------------------------------------
# Usage of the developed optimizer.py model
# -------------------------------------------------------------
from optimizer import *
parameters = {}
parameters['lambda'] = Lambda
parameters['r_p'] = r_p
parameters['P_E'] = P_E
parameters['N_E'] = N_E
parameters['T_E'] = T_E
parameters['T_E_distr'] = T_E_distr
parameters['B_p'] = B_p
parameters['C_E'] = C_E
parameters['N_C'] = N_C
parameters['T_C'] = T_C
parameters['T_C_distr'] = T_C_distr
parameters['C_C'] = C_C
parameters['C_C_pricing'] = C_C_pricing
parameters['W_cr'] = W_cr
parameters['T_bat_cr'] = T_bat_cr
tstart = time.time()
optimized_parameters = find_optimal_configuration(parameters)
tfinish= time.time()
t_optimizer = tfinish-tstart
print(60*"-")
print("Results of optimizer.py model:")
print("Execution time = ",t_optimizer,"seconds")
# print(optimized_parameters)
print(f"Optimal NE: {int(optimized_parameters['N_E_opt'])}")
print(f"Optimal NC: {int(optimized_parameters['N_C_opt'])}")
print(60*"=")
print("optimizer.py is %.2f times faster"%(t_scipy_optimize/t_optimizer))
```

```
Optimization terminated successfully.
         Current function value: 94.730013
         Iterations: 1
         Function evaluations: 68
------------------------------------------------------------
Results of scipy.optimize:
Execution time =  0.008230209350585938 seconds
Optimal NE: 59
Optimal NC: 27
------------------------------------------------------------
Results of optimizer.py model:
Execution time =  0.0007421970367431641 seconds
Optimal NE: 59
Optimal NC: 27
============================================================
optimizer.py is 11.09 times faster
```