**ΕΡΓΑΣΙΑ ΚΑΤΑΝΕΜΗΜΕΝΑ ΣΥΣΤΗΜΑΤΑ**

ΜΙΧΑΛΗΣ ΣΟΛΟΜΩΝΙΔΗΣ

ΑΜ 2121063

#ΤΟ ΠΡΟΓΡΑΜΜΑ ΕΙΝΑΙ ΓΡΑΜΜΕΝΟ ΣΕ GOOGLE COLAB

#Η ΕΡΓΑΣΙΑ ΕΙΝΑΙ ΠΑΝΩ ΣΤΟ WORKFLOW SCHEDULE

import networkx as nx #δημιουργια και διαχειριση γραφηματων

import numpy as np #για αριθμητικες πραξεις με πινακες/διανυσματα

from scipy.optimize import minimize #για βελτιστοποιηση

# Δημιουργία του γράφου επεξεργαστών

G\_processors = nx.Graph()

# Προσθήκη επεξεργαστών με χαρακτηριστικά cpu και energy

G\_processors.add\_node(1, cpu=8, energy=50)

G\_processors.add\_node(2, cpu=4, energy=30)

G\_processors.add\_edge(1, 2, bandwidth=100)

#Δημιουργία του DAG(κατευθυνομενος ακυκλικος γραφος)τωνεργασιών

G\_tasks = nx.DiGraph()

# Προσθήκη εργασιών με απαιτήσεις CPU

G\_tasks.add\_node(1, cpu\_required=2)

G\_tasks.add\_node(2, cpu\_required=3)

G\_tasks.add\_edge(1, 2, data=50) #ακμη μεταξυ 1 και 2

# Επαλήθευση της δομής των γραφημάτων

assert nx.is\_directed\_acyclic\_graph(G\_tasks), "The task graph must be a DAG"

# Συνάρτηση χρόνου εκτέλεσης

def evaluate\_time\_assignment(assignment):

total\_time = 0

for task, processor in assignment.items():

cpu\_capacity = G\_processors.nodes[processor]['cpu']

task\_cpu\_required = G\_tasks.nodes[task]['cpu\_required']

total\_time += task\_cpu\_required / cpu\_capacity

return total\_time

# Συνάρτηση κατανάλωσης ενέργειας

def evaluate\_energy\_assignment(assignment):

total\_energy = 0

for task, processor in assignment.items():

energy = G\_processors.nodes[processor]['energy']

total\_energy += energy

return total\_energy

# Συνάρτηση κόστους επικοινωνίας

def evaluate\_communication\_cost(assignment):

total\_cost = 0

#καθε ακμη αντιπροσωπευει εξαρτηση δεδομενων μεταξυ 2 εργασιων

for u, v in G\_tasks.edges():

if assignment[u] != assignment[v]:

bandwidth = G\_processors[assignment[u]][assignment[v]]['bandwidth']

data = G\_tasks[u][v]['data']

total\_cost += data / bandwidth

return total\_cost

# Συνολική συνάρτηση κόστους

def total\_cost(assignment):

assignment\_dict = {i+1: int(assignment[i]) for i in range(len(assignment))}

return (

evaluate\_time\_assignment(assignment\_dict) +

evaluate\_energy\_assignment(assignment\_dict) +

evaluate\_communication\_cost(assignment\_dict)

)

# Εκχώρηση

initial\_assignment = [1, 1] # Εκχώρηση εργασιών σε επεξεργαστές

# Περιορισμοί (διασφαλίζει ότι η χρήση CPU κάθε επεξεργαστή δεν υπερβαίνει τη διαθέσιμη CPU του επεξεργαστή αυτού.)

def constraint\_cpu\_capacity(assignment):

assignment\_dict = {i+1: int(assignment[i]) for i in range(len(assignment))}

cpu\_usage = {proc: 0 for proc in G\_processors.nodes}

for task, proc in assignment\_dict.items():

cpu\_usage[proc] += G\_tasks.nodes[task]['cpu\_required']

return [G\_processors.nodes[proc]['cpu'] - usage for proc, usage in cpu\_usage.items()]

# Ορισμός περιορισμών ως λίστα

constraints = [{'type': 'ineq', 'fun': constraint\_cpu\_capacity}]

# Χρήση minimize για τη βελτιστοποίηση

result = minimize(total\_cost, initial\_assignment, method='COBYLA', constraints=constraints)

# Μετατροπή αποτελέσματος σε dictionary για ευκολία ανάγνωσης

optimal\_assignment = {i+1: int(result.x[i]) for i in range(len(result.x))}

print(f'Optimal assignment: {optimal\_assignment}')

print(f'Minimized cost: {result.fun}')

#Η χρηση του for γινεται για να διατρεξει καθε εργασια αλλα και τον αντιστοιχο επεξεργαστη στον οποιον εχει εκχωρηθει οπως περιγραφονται στο assignment--> λεξικο οπου τα κλειδια, οι εργασιες και οι τιμες ειναι επεξεργαστες.

#Σε αυτον τον κωδικα αναθετω εργασιες σε επεξεργαστες λαμβανοντας υποψη τον χρονο εκτελεσης, την καταναλωση ενεργειας αλλα και το κοστος επικοινωνιας. Κανω χρηση του minimize για να βρω την εκχωρηση που θα ελαχιστοποιησει το συνολικο κοστος αλλα ταυτοχρονα να μην υπερβαινει η χρηση CPU την διαθεσιμη CPU.

#H μετατροπη του assignment σε λεξικο γινεται ωστε τα κλειδια να ειναι οι εγρασιες και οι τιμες οι επεξεργαστες που εχουν εκχωρηθει οι εργασιες. Για να ταιριαξει η δομη δεδομενων με τις αλλες συναρτησεις αξιολογησης που παιρνουν λεξικο ως εισοδο.