[Date]

Student Name

University name

Fog Computing

# Table of Contents

**Introduction1**

Background2

Overview of Fog Computing3

Objectives 3

Problem Statement3

**Type chapter title (level 1)4**

Type chapter title (level 2)5

Type chapter title (level 3)6

**Abstract**

Fog computing is used in conjunction with cloud computing to meet the growing demand for IT-related services. With the rapid growth of technology, fog computing is developing as a promising way for converting and retrieving data from Internet of Things applications. This essay discusses the concept, design, and application of fog computing in both new and existing applications.

Fog machines come in many shapes and sizes. One of the most significant challenges to running an IoT application in the fog is resource allocation. This study will examine

Contemporary research on resource allocation.

# **Background**

Fog computing, often called edge computing, is a type of decentralized computing infrastructure where applications, data, and compute are located close to the data source, typically at the edge of the network. This method places computer resources closer to the locations where data is generated and consumed in an effort to circumvent the limitations of conventional cloud computing infrastructures.

The growing number of Internet of Things (IoT) devices and the growing demand for real-time and low-latency applications gave rise to the concept of fog computing. Even though it is robust and scalable, traditional cloud computing frequently encounters latency and bandwidth issues when dealing with the large amounts of data generated by Internet of Things devices. By shifting compute closer to the network's edge, fog computing improves scalability, reduces latency, speeds up reaction times, and increases bandwidth efficiency.

# **Introduction**

IoT environments are made up of loosely connected devices connected via heterogeneous networks. Building such environments involves gathering and analysing data from IoT devices to identify patterns, do predictive analysis, and make timely decisions. Data in such environment is of two types:

|  |  |
| --- | --- |
| Little Data or Big Stream | Big Data |
| Transient data that is captured constantly by IoT devices. | **Persistent data and knowledge stored and archived in centralized cloud storage.** |

Fog computing is a distributed computing concept that extends cloud services to network edges (see Figure 1.1). It enables efficient management and programming of computation, networking, and storage services between data centres and end devices. Fog computing combines cloud and edge devices, such as smart gateways, routers, and dedicated fog devices, to run application components.

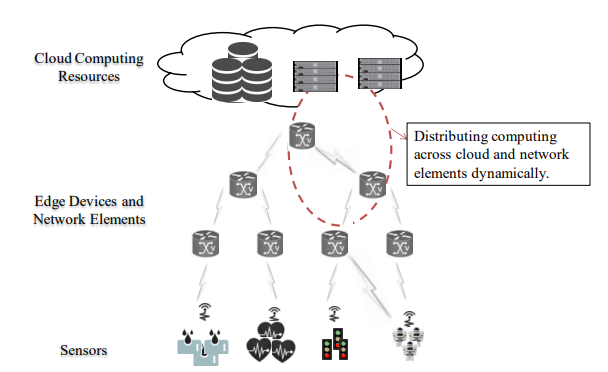


Figure 1.1: Fog Extends Cloud to Edge

Fog Computing focused on platform specification application and technology.

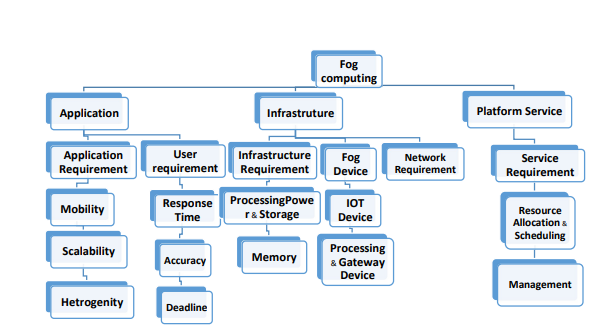


Figure 1.1.1: Taxonomy of Fog Computing

# **Architecture of Fog Computing**

The Fog Computing architecture consists of three Layers named as IoT layer, fog layer and cloud layer respectively.

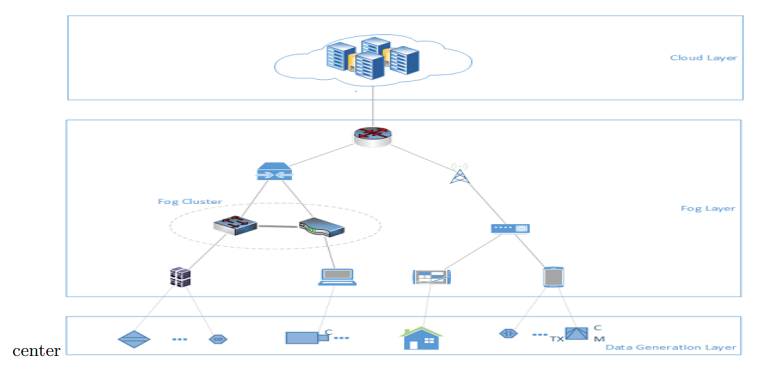


Figure 2: Architecture of fog computing

End Devices layer

It is the final layer of architecture closest to the physical domain. These devices include smartphones, laptops, smart grids, smart cards, and smart household equipment. They are responsible for sensing, transmitting, and processing data from physical entities or events to higher-level systems.

Fog Layer

This layer contains all fog gadgets. Fog devices include intelligent network devices such as routers and switches. Terminal devices can quickly connect to fog devices in the nearby network. Fog devices can process, store, and analyse data.

Cloud Layer

This layer consists of remote cloud data centres, powerful servers, and devices. This layer effectively manages the underlying layer using sophisticated devices.

# **Problem Statement**

# **Optimization Problem**

Optimize multi conflicting objectives defined by MOP2 function and subject to decision variables within the range -4 to 4. This function will assess solutions based on two objectives. The goal is to optimize a set of choice variables in order to minimize/maximize several objectives at the same time.

* Minimize z1 = 1 – exp (-sum(x-1/sqrt(n)\*\*2))
* Minimize z2 = 1 – exp (-sum(x+1/sqrt(n)\*\*2))

**Decision variables are:**

* %=5 variables (nVar =5)
* Each variable lies within the range of -4 to 4 (varMin = -4, var Max= 4)

Use Multi-objective Particle Swarm Optimization (MOPSO) algorithm to solve above problem and use a grid based approach for the selection of a leader and diversity maintenance.

Let us explain what Multi-Objective Particle Swarm Optimization is:

**Multi-Objective Particle Swarm Optimization**

To address the inadequacies of particle swarm optimization (PSO) in handling multiobjective optimization problems, a multiobjective particle swarm optimization (MOPSO) approach is presented. Many engineering challenges are made up of numerous goals that interact and conflict with one another. When tackling practical problems, people frequently confront many objectives that must be met simultaneously in order to reach the best optimal solution, also known as multiobjective optimization problems. The optimization issue has many optimization objectives and must be solved concurrently, resulting in a multiobjective optimization problem (MOP). To solve the multiobjective optimization problem, the following three main factors must be addressed:

1. The solution set comes as close to the Pareto front as possible.
2. Maintain a high level of demographic variety.
3. Ensure that the particles are effectively and uniformly distributed in the solution space.

A heuristic swarm intelligence system called particle swarm optimization (PSO) uses bird swarm behaviour as a model to solve optimization problems. The algorithm provides outstanding flexibility and stability. A number of specialists have directed their attention toward an intelligent global optimization algorithm in recent years. With its unique position and speed, every particle in the PSO algorithm is comparable to a bird in the population. To locate the global optimum solution, the particles use social learning and self-learning to navigate around the solution space. The following update formula for the particle's location and velocity applies, assuming that the particle's population size is and the space's dimension is:

(p)+ ,

Is determined as;

Where the maximum and minimum values of inertia weight and Tmax are is the maximum number of iterations.

* 1. **Implementation of PSO Algorithm in Python**

|  |
| --- |
| 1. import numpy as np 2. from numpy import matlib 3. import matplotlib.pyplot as plt 4. import random as random 5. import math 6. def deleteOneRepositoryMember(rep , gamma): 7. gridindices = [item.gridIndex for item in rep] 8. OCells = np.unique(gridindices) # ocupied cells 9. N = np.zeros(len(OCells)) 10. for k in range(len(OCells)): 11. N[k] = gridindices.count(OCells[k]) 12. # selection probablity 13. p = [math.exp(gamma\*item) for item in N] 14. p = np.array(p)/sum(p) 15. # select cell index 16. sci = roulettewheelSelection(p) 17. SelectedCell = OCells[sci] 18. #selected Cell members 19. selectedCellmembers = [item for item in gridindices if item == SelectedCell] 20. selectedmemberindex = np.random.randint(0,len(selectedCellmembers)) 21. #selectedmember = selectedCellmembers[selectedmemberindex] 22. # delete memeber 23. #rep[selectedmemberindex] = [] 24. rep = np.delete(rep, selectedmemberindex) 25. return rep.tolist() 26. def SelectLeader(rep , beta): 27. gridindices = [item.gridIndex for item in rep] 28. OCells = np.unique(gridindices) # ocupied cells 29. N = np.zeros(len(OCells)) 30. for k in range(len(OCells)): 31. N[k] = gridindices.count(OCells[k]) 32. # selection probablity 33. p = [math.exp(-beta\*item) for item in N] 34. p = np.array(p)/sum(p) 35. # select cell index 36. sci = roulettewheelSelection(p) 37. SelectedCell = OCells[sci] 38. #selected Cell members 39. selectedCellmembers = [item for item in gridindices if item == SelectedCell] 40. selectedmemberindex = np.random.randint(0,len(selectedCellmembers)) 41. # selectedmember = selectedCellmembers[selectedmemberindex] 42. return rep[selectedmemberindex] 43. def roulettewheelSelection(p): 44. r = random.random() 45. cumsum = np.cumsum(p) 46. y = (cumsum<r) 47. x= [i for i in y if i==True] 48. return len(x) 49. def FindGridIndex(particle, grid): 50. nObj = len(particle.cost) 51. NGrid = len(grid[0].LowerBounds) 53. particle.gridSubIndex = np.zeros((1,nObj))[0] 54. for j in range(nObj): 55. index\_in\_Dim = len( [item for item in grid[j].UpperBounds if particle.cost[j]>item]) 56. particle.gridSubIndex[j] = index\_in\_Dim 57. particle.gridIndex = particle.gridSubIndex[0] 58. for j in range(1,nObj): 59. particle.gridIndex = particle.gridIndex 60. particle.gridIndex = NGrid\*particle.gridIndex 61. particle.gridIndex = particle.gridIndex + particle.gridSubIndex[j] 62. return particle 63. def CreateGrid(pop,nGrid,alpha,nobj): 64. costs = [item.cost for item in pop] 65. Cmin = np.min(costs,axis=0) 66. Cmax = np.max(costs,axis=0) 67. deltaC = Cmax - Cmin 68. Cmin = Cmin - alpha\*deltaC 69. Cmax = Cmax + alpha\*deltaC 71. grid = [GridDim() for p in range(nobj)] 72. for i in range(nobj): 73. dimValues = np.linspace(Cmin[i],Cmax[i],nGrid+1).tolist() 74. grid[i].LowerBounds = [-float('inf')] + dimValues 75. grid[i].UpperBounds = dimValues + [float('inf')] 76. return grid 77. def Dominates(x,y): 78. x=np.array(x) 79. y=np.array(y) 80. x\_dominate\_y = all(x<=y) and any(x<y) 81. return x\_dominate\_y 82. def DetermineDomination(pop): 83. pop\_len= len(pop) 84. for i in range(pop\_len): 85. pop[i].IsDominated = False 86. for i in range(pop\_len-1): 87. for j in range(i+1,pop\_len): 88. if Dominates(pop[i].cost,pop[j].cost): 89. pop[j].IsDominated = True 90. if Dominates(pop[j].cost,pop[i].cost): 91. pop[i].IsDominated = True 92. return pop 94. # problem definition 95. def MOP2(x): 96. x = np.array(x) 97. n= len(x) 98. z1 = 1 - math.exp(-sum((x-1/math.sqrt(n))\*\*2)) 99. z2 = 1 - math.exp(-sum((x+1/math.sqrt(n))\*\*2)) 100. return [z1,z2] 101. costfunction = lambda x: MOP2(x) 102. nVar = 5 # number of decision vars 103. varMin = -4 104. varMax = 4 105. maxIt = 100 106. nPop = 200 # population size 107. nRep = 50 # size of repository 108. w = 0.5 # inertia wieght 109. c1 = 2 # personal learning coefficient 110. c2 = 2 # global learning coefficient 111. wdamping = 0.99 112. # ################ constriction coefficients 113. # phi1 = 2.05 114. # phi2 = 2.05 115. # phi = phi1+phi2 116. # chi = 2/(phi - 2 + np.sqrt(phi\*\*2 - 4\*phi)) 117. # w = chi # inertia wieght 118. # c1 = chi\*phi1 # personal learning coefficient 119. # c2 = chi\*phi2 # global learning coefficient 120. # wdamping = 1 121. # ################# 122. beta = 1 # leader selection pressure 123. gamma = 1 # deletion selection pressure 124. NoGrid = 5 125. alpha=0.1 # nerkhe tavarrom grid 126. # initialization 127. class Particle: 128. position = [] 129. cost = [] 130. velocity = [] 131. best\_position = [] 132. best\_cost = [] 133. IsDominated = [] 134. gridIndex = [] 135. gridSubIndex = [] 136. # for each objective a grid items is division of values of objective cost 137. class GridDim: 138. LowerBounds = [] 139. UpperBounds = [] 140. #Particles = np.matlib.repmat(Particle,nPop,1) 141. Particles = [Particle() for p in range(nPop)] 142. for i in range(nPop): 143. Particles[i].position = np.random.uniform(varMin,varMax,nVar) 144. Particles[i].velocity = np.zeros(nVar) 145. Particles[i].cost = costfunction(Particles[i].position) 146. # update best personal Best 147. Particles[i].best\_position = Particles[i].position 148. Particles[i].best\_cost = Particles[i].cost 149. Particles[i].IsDominated = False 150. Particles = DetermineDomination(Particles) 151. Repos = [item for item in Particles if item.IsDominated == False ] 152. nObj =len( Repos[0].cost) 153. grid = CreateGrid(Repos,NoGrid,alpha=0.1,nobj=nObj) 154. for r in range(len(Repos)): 155. Repos[r] = FindGridIndex(Repos[0],grid) 156. # MOPSO main loop 157. for it in range(maxIt): 158. for i in range(nPop): 159. leader = SelectLeader(Repos,beta) 160. # update velocity 161. Particles[i].velocity = w\*Particles[i].velocity \ 162. + c1\*np.random.rand(1,nVar)[0]\*(Particles[i].best\_position - Particles[i].position) \ 163. + c2\*np.random.rand(1,nVar)[0]\*(leader.position - Particles[i].position) 165. # update position 166. Particles[i].position = Particles[i].position + Particles[i].velocity 167. # evaluation 168. Particles[i].cost = costfunction(Particles[i].position) 169. if Dominates(Particles[i].cost,Particles[i].best\_cost): 170. Particles[i].best\_position = Particles[i].position 171. Particles[i].best\_cost = Particles[i].cost 172. else: 173. if np.random.rand() > 0.5: 174. Particles[i].best\_position = Particles[i].position 175. Particles[i].best\_cost = Particles[i].cost 177. Repos = Repos + Particles 178. Repos = DetermineDomination(Repos) 179. Repos = [item for item in Repos if item.IsDominated == False ] 180. grid = CreateGrid(Repos,NoGrid,alpha=0.1,nobj=nObj) 181. for r in range(len(Repos)): 182. Repos[r] = FindGridIndex(Repos[r],grid) 183. # check if repository is full 184. if len(Repos) > nRep : 185. extra = len(Repos) - nRep 186. for e in range(extra): 187. Repos = deleteOneRepositoryMember(Repos,gamma) 188. ########## show figure ########## 189. plt.clf() 190. particlesCost = np.reshape( [item.cost for item in Particles ],newshape=(nPop,2)) 191. repositoryCost = [item.cost for item in Repos] 192. repositoryCost = np.reshape( repositoryCost, newshape=(len(repositoryCost),2)) 193. plt.plot(particlesCost[:,0], particlesCost[:,1], 'o' ,mfc='none') 194. plt.plot(repositoryCost[:,0], repositoryCost[:,1], 'r\*') 196. plt.draw() 197. plt.pause(0.00000000001) 198. w=w\*wdamping 200. # print(repositoryCost) 201. # print("ok") 202. # print(particlesCost) 203. ########## show figure ########## 204. plt.show() |

* 1. **Mathematical and Theoretical Formulation**

Multi-Objective Particle Swarm Optimization (MOPSO) is a metaheuristic optimization approach designed to tackle problems with many competing objectives. It iteratively searches the solution space, learning from the collective behaviour of fish schools and bird flocks. Within MOPSO, a group of potential solutions, known as particles, wander across the search space, adjusting their positions in response to both their personal experiences and the information from their surroundings. Unlike traditional Particle Swarm Optimization (PSO), MOPSO focuses on maximizing many objectives at simultaneously, with the goal of identifying a set of solutions known as the Pareto front that represent trade-offs between these objectives. Keeping a store of non-dominated solutions ensures that the algorithm evaluates a diverse range of great responses. MOPSO solves multi-objective optimization problems in a stable and scalable manner by efficiently navigating the solution space via iterative refinement and selection techniques.

* 1. **Efficient Implementation of PSO Algorithm**

* Particle Swarm Optimization (PSO) is an effective technique for solving the optimization problem. In discrete and continuous environments, the PSO method can tackle single- and multi-objective optimization problems. The following are the implementation's key features:

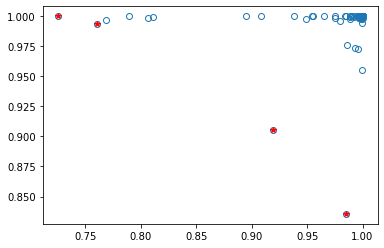
|  |
| --- |
| Iterative adjustments to particle locations and velocities are performed using equations derived from Multi-Objective Particle Swarm Optimization (MOPSO). |
| To improve the search experience, the algorithm considers additional charges and adjusts personal best placements as needed. |
| The repository's composition changes dynamically, resulting in grid partitions.  Grid indices for repository members are updated to ensure accurate representation in the goal space. |
| Scatter plots of the present population and repository solutions provide real-time visualization. |
| Regularly updating the display makes it easy to find convergence trends and track the optimization process. |

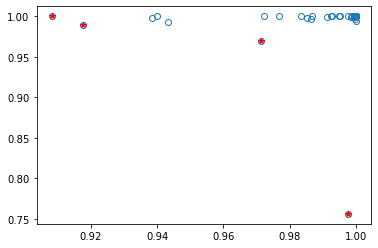
* 1. **Optimization Loop**
* For a predetermined number of iterations (maxIt), the core optimization loop iterates. With each iteration:   
    
  Particle positions and speeds are adjusted for effective solution space exploration.   
  In order to appropriately conserve non-dominated solutions, the repository is updated and maintained on a regular basis.   
  To guarantee that the solution space is best represented, grid partitions are changed.   
  To give information about the continuous optimization process, visualization elements are updated on a regular basis.  
  1. **Parameters**

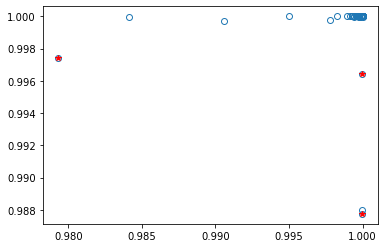
The number of decision variables (nVar), population size (nPop), repository size (nRep), and selection pressure parameters (beta, gamma) are all factors that influence the algorithm's behavior and performance. These variables influence how exploration and exploitation are balanced during the optimization process, which influences the quality and diversity of solutions produced.

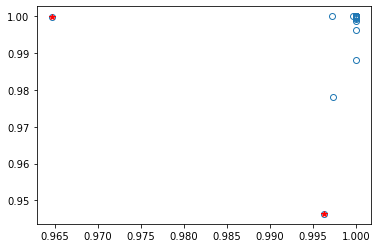
4 Simulation and Results

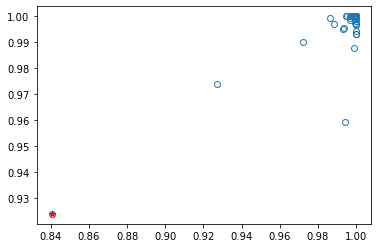
The algorithm is simulated using anaconda the spyder. We got following graphs after running this.

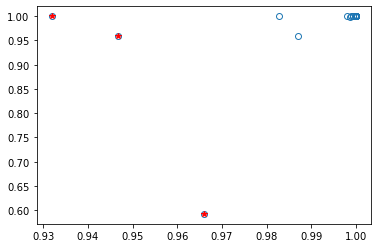


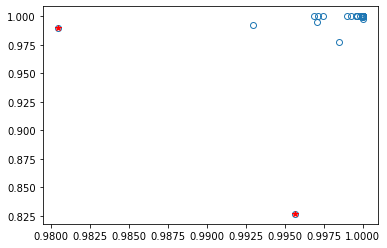


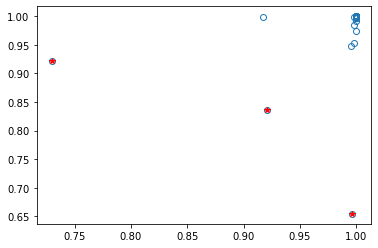


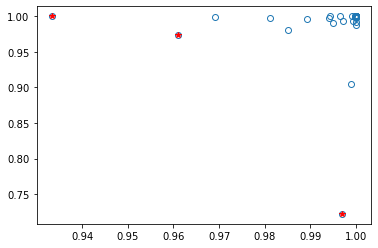
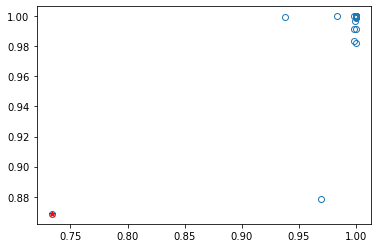
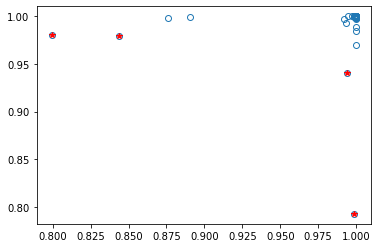
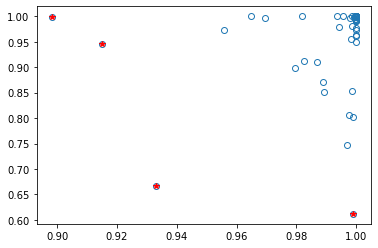
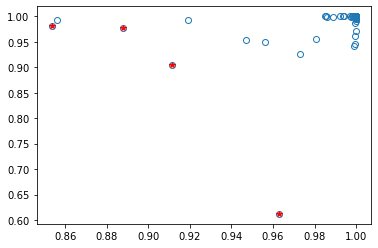


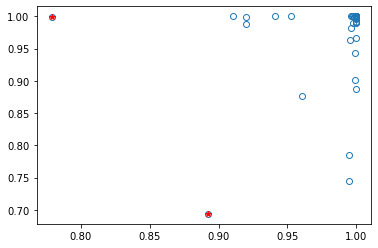
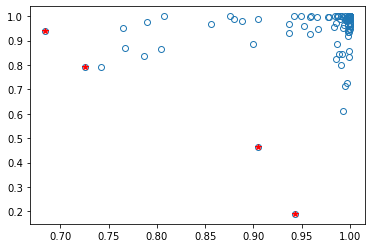
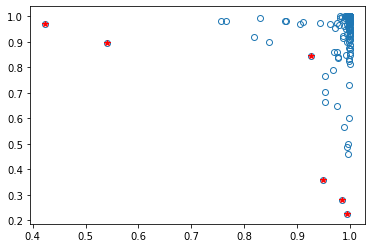
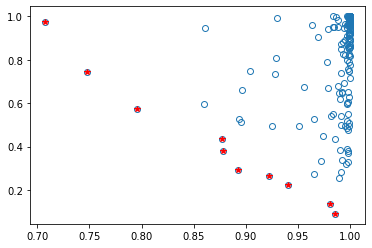


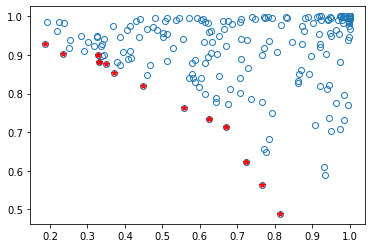








**4.1 Creation of a Random Network**

import numpy as np

import networkx as nx

import matplotlib.pyplot as plt

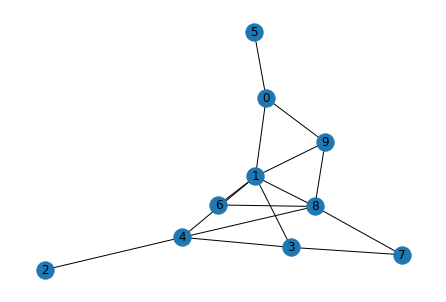
# Generate a random Erdős-Rényi graph with 10 nodes and probability 0.3

G = nx.erdos\_renyi\_graph(10, 0.3)

# Draw the graph

nx.draw(G, with\_labels=True)

plt.show()



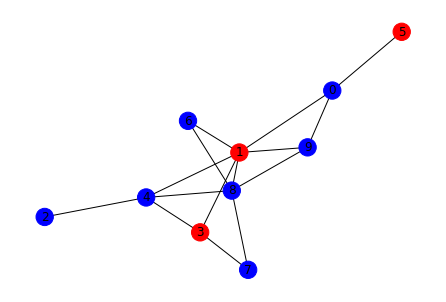
infected\_nodes = [1, 3, 5]

# Example of updating node attributes and re-visualizing the graph

node\_colors = ['r' if node in infected\_nodes else 'b' for node in G.nodes()]

nx.draw(G, with\_labels=True, node\_color=node\_colors)

plt.show()



Conclusion

Finally, the use of Multi-Objective Particle Swarm Optimization (MOPSO) in fog computing offers a potential approach to addressing the complex optimization difficulties inherent in fog-based systems. MOPSO's capacity to simultaneously maximize many competing objectives enables efficient resource allocation, energy management, and service placement in fog situations. MOPSO uses particle representation, grid division, and repository management techniques to develop Pareto-optimal solutions that balance performance parameters including latency, energy consumption, and cost. Real-time visualization improves knowledge and decision-making in fog computing environments.

Future Work

As fog computing evolves and plays an increasingly important role in distributed computing architectures, future research can focus on developing MOPSO-based optimization approaches suited specifically for fog environments. Possible possibilities for future research include:   
  
Dynamic Resource Allocation: Creating adaptive MOPSO algorithms that can dynamically distribute resources in fog environments based on changing workloads, network conditions, and user demands.   
Energy-Aware Optimization: Looking at MOPSO variants that prioritize energy-efficient resource allocation and job scheduling in order to extend the battery life of edge devices and reduce carbon footprint.

Security and Privacy Considerations: Incorporating security and privacy-conscious objectives into the optimization process to reduce risks to data confidentiality, integrity, and availability in fog computing systems.   
Edge Intelligence and Machine Learning: Investigating the use of machine learning techniques in MOPSO to enable intelligent edge decision-making, such as predictive resource provisioning, anomaly detection, and adaptive task offloading.   
Multi-Objective Service Placement: Extending MOPSO to solve multi-objective service placement difficulties, taking into account factors such as service availability, reliability, and Quality of Service (QoS) in fog situations.

Scalability and Edge Device Constraints: Investigating scalable MOPSO variations that are tailored for resource-constrained edge devices, taking into account restrictions in processor power, memory, and communication bandwidth.   
Validation and benchmarking: Extensive empirical research and simulations will be conducted to validate the effectiveness and performance of MOPSO-based optimization approaches in real-world fog computing scenarios, with a comparison to existing optimization methods and benchmarks.