Deep reinforcement learning-based optimal deployment of IoT machine learning jobs in fog computing architecture

Omid Bushehrian, Amir Moazeni

By: Mohammad Kadkhodaei

1404-04-07, rev 1.0

Abstract

- Many IoT applications are machine learning jobs that collect and analyze sensor measurements.
- These machine learning jobs consist of distributed tasks that work collaboratively to build models in a federated manner.
- The problem of determining the optimal number and the coverage of distributed tasks of an IoT machine learning job has not been addressed previously
- This paper proposes a two-phased method for adaptive task creation and deployment of IoT ML jobs over a heterogeneous multi-layer fog computing architecture (adaptively and efficiently).

Abstract

- In the 1st phase, the optimal number of tasks and their respective sensor coverage is determined using a Deep Reinforcement Learning (DRL) > Task Creation, using > Deep Deterministic Policy Gradient (DDPG)
- In the 2nd phase, the tasks are deployed over the heterogeneous multi-layer fog computing architecture using a greedy deployment method > Task Deployment
- DDPG Sensor positioning Component + Task creation Component + Greedy Deployment Component
- The task creation and deployment is formulated as a 3- optimization problem:
- 1) min the deployment latency
- 2) min the deployment cost
- 3) min the evaluation loss

Result

- The proposed two-phased DRL-based method could outperform the Edge-IoT and Cloud-IoT baseline methods by improving the total deployment score up to 32%.

$$Maximize \sum_{j_i \in J} deploymentScore_j$$

$$deploymentScore_j = -1 \times (w_1 \times latencyFactor_j^N + w_2 \times costFactor_j^N + w_3 \times lossFactor_j^N)$$

- Simulation: Python + Keras/TensorFlow, COREi7, 32MB RAM

Table 1 The comparison of the related works using different criteria. Time, cost, accuracy, resource utilization, and energy consumption are denoted by T, C, A, R, and E respectively in the objectives column

Paper	Category	FL-based	Application architecture	Task creation	Infrastructure	Algorithm	Objectives				
							T	C	A	R	E
[3]	Mathematical/(meta-) heuristic methods	No	Microservice	No	Mobile edge	Branch-and- bound	×	●	×	×	×
[20]		No	DAG	No	Multi-layer Fog	Dynamic scheduling	●	×	×	×	×
[21]		No	Microservice	No	Fog	PSO	●	●	×	×	×
[22]		No	Independent multi-task	No	Fog	AEO	●	×	×	×	×
[23]		No	Independent multi-task	No	Fog	GWO	●	●	×	×	×
[24]		No	Independent services	No	Fog	WOA	●	×	×	×	●
[25]		No	Worker/parameter server	Yes(non-adaptive)	Cluster	Approximation algorithms	●	×	×	×	×
[4]	DRL-based	Yes	FL- three-layer Architecture	Yes (fixed task number)	Edge/Cloud	MDRL	●	×	●	×	●
[10]		No	DAG	No	Heterogeneous fog	DDRL	●	×	×	×	●
[26]		No	DAG	No	Fog	RL	●	●	×	×	×
[27]		No	Independent multi-task	No	Mobile edge	DDLO	●	×	×	●	●
[28]		No	Independent multi-task	No	Edge	DRL	●	×	×	●	×
[29]		No	Independent multi-task	No	Edge	DQN	●	×	×	●	×
[30]		No	Microservice	No	Heterogeneous fog	A3C	●	●	×	×	×
[31]		No	Cloud jobs	No	Cloud	Actor-Critic DRL	●	×	×	×	●
[32]		No	PoW single task	No	MCS	DRL	●	×	×	×	●
[33]		No	C-PoW single task	No	MCS	DRL	●	×	×	×	●
[34]		No	Single task	No	MEC	DRL	×	●	×	●	×
Proposed method	Ensemble	Yes	FL- two-layer architecture	Yes (adaptive)	Multi-layer heterogeneous Fog	DDPG-Greedy	●	●	●	×	×

$$Maximize \sum_{j_j \in J} deploymentScore_j$$

$$deploymentScore_j = -1 \times (w_1 \times latencyFactor_j^N + w_2 \times costFactor_j^N + w_3 \times lossFactor_j^N)$$

$$latencyFactor_j = \sum_{h_k \in F} \sum_{s_i \in M_j} X_{i,j,k} \times L_k$$

$$costFactor_j = compCost_j + commCost_j$$

$$lossFactor_j = \left| T_j \right|$$

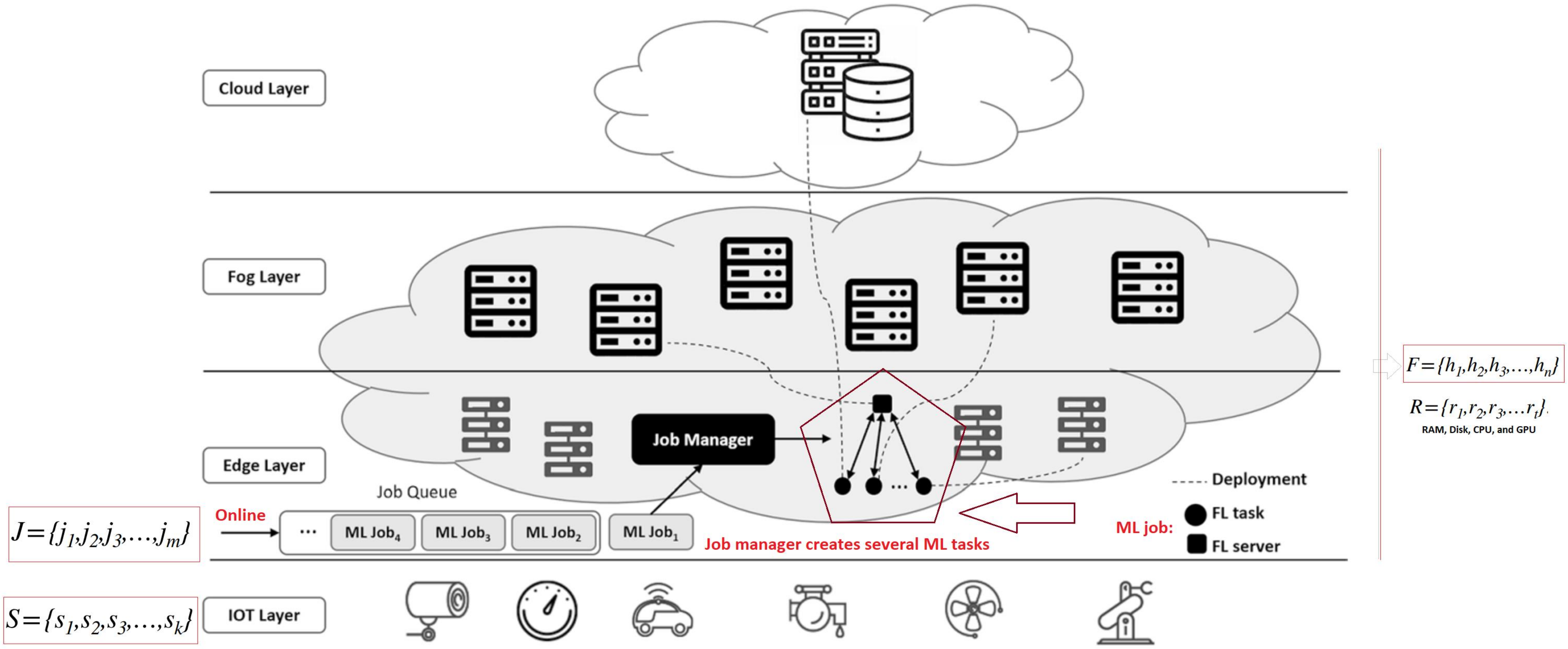
Jobs, Job Manager, Tasks set

- Both offline and online deployment problem are NP-Hard
- Jobs are received by the job manager over time and the job manager makes the deployment once a new job is received.

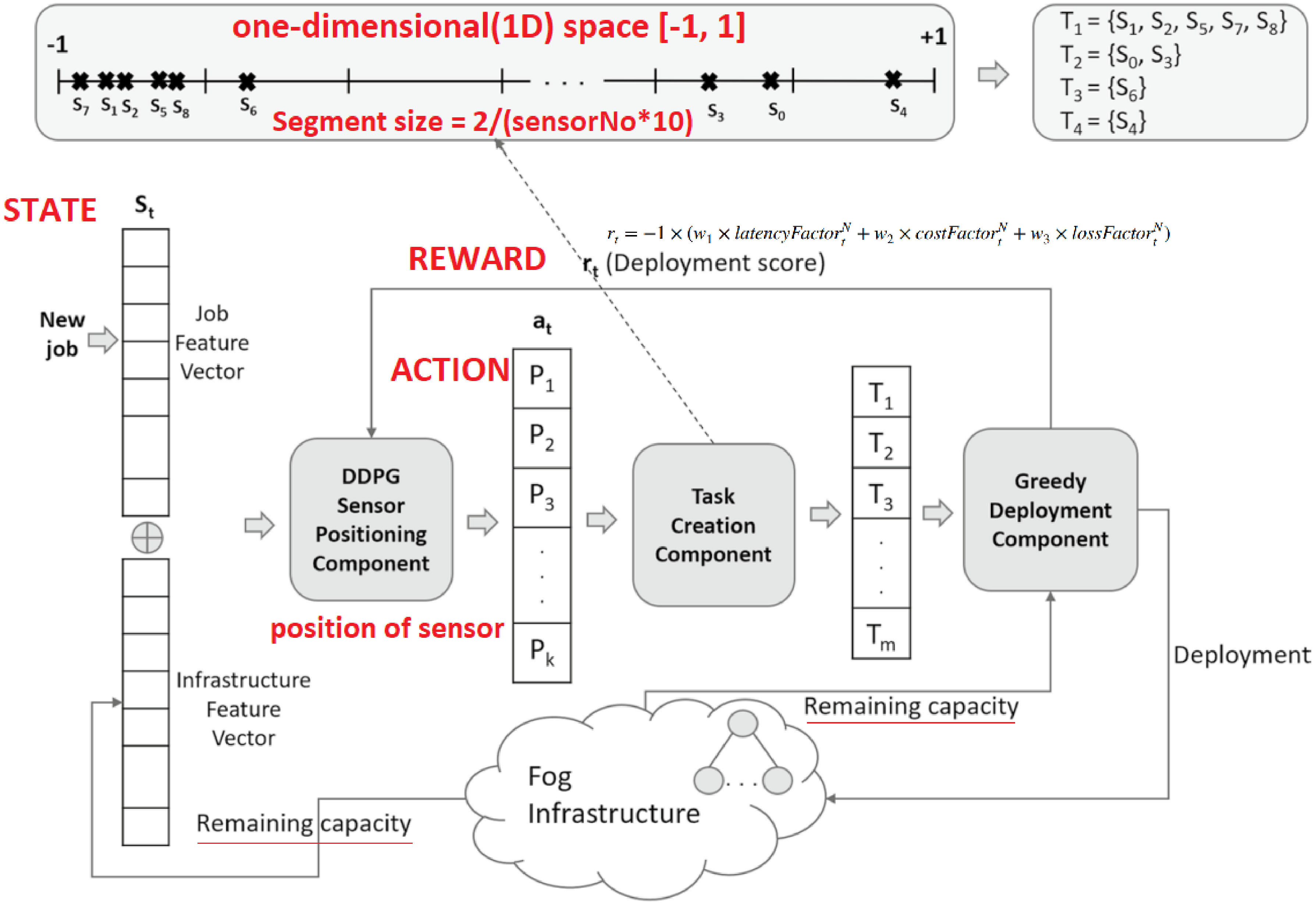
$$\text{IoT ML jobs, } J = \{j_1, j_2, j_3, \ldots, j_m\}$$

the aim is to define the

task set $T_j = \left\{ t_k^{n,j} \middle| h_k \in F \; and \; 1 \leq n \leq g_j \right\}$ for job j_j



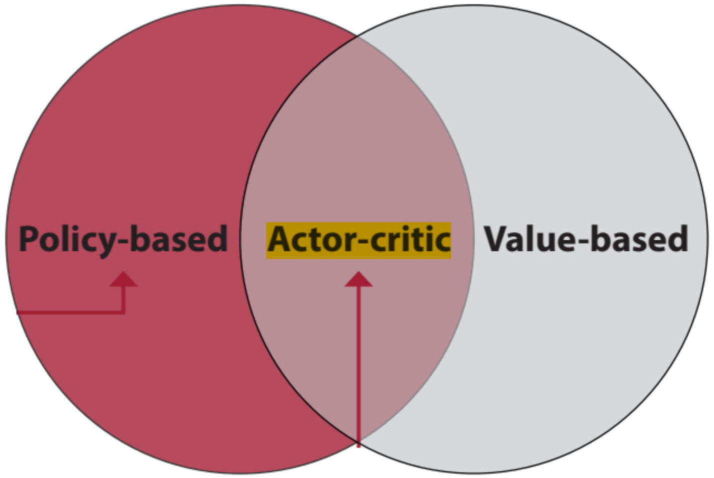
Cloud Layer
Fog Layer
Edge Layer
IOT Layer
Job Manager
Job Queue
Online
$J=\{j_1,j_2,j_3,\ldots,j_m\}$
$S=\{s_1,s_2,s_3,\ldots,s_k\}$
$F=\{h_1,h_2,h_3,\ldots,h_n\}$
$R=\{r_1,r_2,r_3,\ldots r_t\}$
RAM, Disk, CPU, and GPU
... ML Job$_4$ ML Job$_3$ ML Job$_2$ ML Job$_1$
Job manager creates several ML tasks
Deployment
ML job:
FL task
FL server



one-dimensional(1D) space [-1, 1]
-1
+1
S_7 S_1 S_2 S_5 S_8 S_6 S_3 S_0 S_4
Segment size = 2/(sensorNo*10)
$T_1 = \{S_1, S_2, S_5, S_7, S_8\}$
$T_2 = \{S_0, S_3\}$
$T_3 = \{S_6\}$
$T_4 = \{S_4\}$
STATE
S_t
New job
Job Feature Vector
Infrastructure Feature Vector
$r_t = -1 \times (w_1 \times latencyFactor_t^N + w_2 \times costFactor_t^N + w_3 \times lossFactor_t^N)$
REWARD
r_t (Deployment score)
a_t
ACTION
P_1
P_2
P_3
P_k
DDPG Sensor Positioning Component
position of sensor
Task Creation Component
T_1
T_2
T_3
T_m
Greedy Deployment Component
Deployment
Remaining capacity
Fog Infrastructure
Remaining capacity

DRL

- 1) Value-based methods : Q-learning, SARSA, DQN, and company
- 2) Policy-based methods : REINFORCE, …
- 3) Actor-Critic, learn both policies and value functions : A3C, DDPG, …



DDPG : Deep Deterministic Policy Gradient

- In 2014, Silver et al. released the deterministic policy gradient (DPG) algorithm
- DDPG was introduced in 2015 in a paper titled “Continuous control with deep reinforcement learning.” The paper was authored by Timothy Lillicrap (et al.) while he was working at Google DeepMind as a research scientist
- Is a popular model-free, off-policy and actor–critic DRL algorithm
- It consists of 4- DNNs: actor, critic, target actor, target critic

Greedy Algorithm

- Iteratively places tasks on the nodes from the large to the small sizes
- Size = total computation demand
- the FL server is deployed first.
- A task deployment score is computed based on the latency, computation and communication costs
- The node with the best deployment score is selected for the task placement

The greedy job deployment algorithm

```
Data: T: FL Tasks, F: Infrastructure nodes
Result: deployment: Task's deployment
foreach task ∈ T do
    demand[task] ← calcDemand(task);
end
while T contains undeployed tasks do
    if FLserver is not deployed then
        selectedTask ← FLserver;                    First FLserver
    else
        selectedTask ← nextLargestTask(demand);     nextLargestTask
    end
    foreach h ∈ F do
        if demand(selectedTask) ≤ remainingCapacity(h) then
            score[h] ← calcScore(h, selectedTask);
        else                                        Check Have Capacity
            score[h] ← −∞;
        end
    end
    selectedNode ← nodeWithMaxScore(score);
    deployment[selectedTask] ← selectedNode;
    Update selectedNode capacity
    Mark selectedTask as deployed
end
```

Federated Learning

- Locally-trained model parameters are received and aggregated by a centralized server to update the global model parameters using a model aggregation algorithm like : Federated averaging (FedAvg)
- Using : Flower framework + (LSTM) neural network in Python
- Dataset: air-quality measurements (7-features)

Future works

1. Replacing the greedy algorithm with a second DRL-based deployment algorithm
2. Support unsupervised ML jobs

My Idea

- Improve Queue structure
- Determine Jobs structure more clearly
- Add priority to jobs
- Use Meta Heuristics instead of Greedy algorithm
- Use another DNN instead of Greedy algorithm
- Redundant job manager + Fault tolerance
- Consider energy consumption