UNIVERSITÀ DI PISA

AI-DII

Hands-on session 2
FedLang: A Middleware Support to Federated Learning Experiments

Distributed Systems and Middleware Technologies

2023/2024

By José Luis Corcuera Bárcena

Agenda

1. A quick overview of Federated Learning (FL)
2. Federated Learning algorithmic approaches
3. Existing FL frameworks
4. FedLang: Overview
5. FedLang: Example of FL experiment execution
6. FedLang: Experimental Setup
7. FedLang: Performance results
8. Pyrlang in Action
9. Monitoring Erlang nodes

A quick overview of Federated Learning (1)

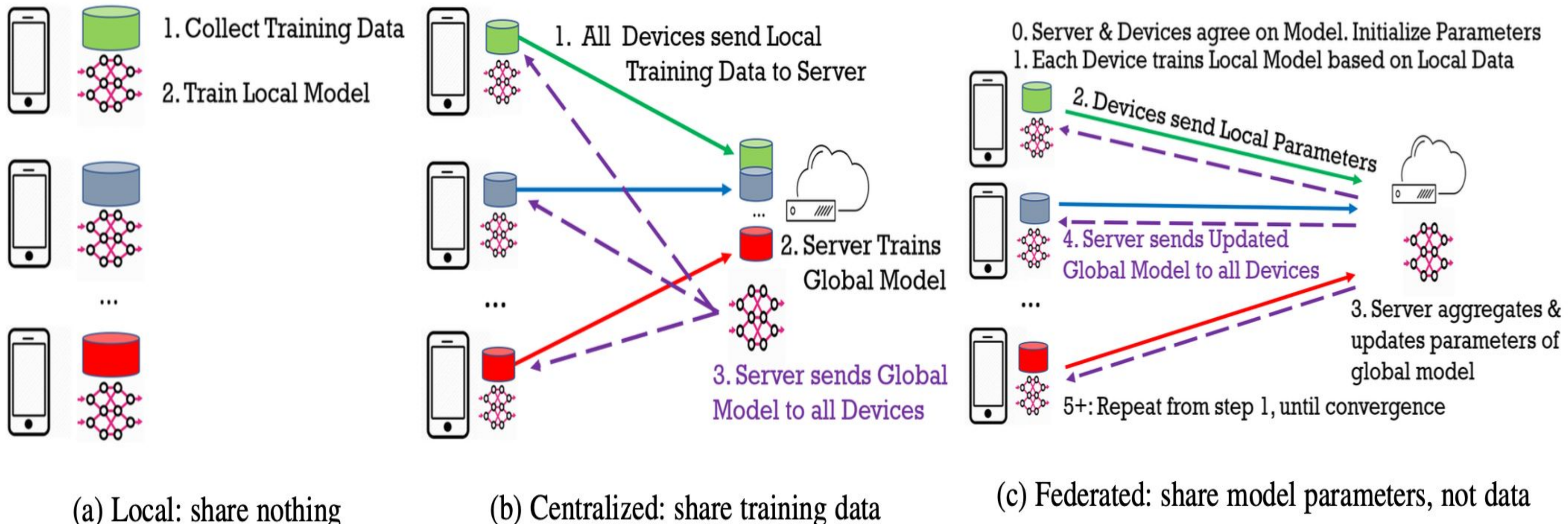


Image extracted from: Bakopoulou, E., Tillman, B., & Markopoulou, A. (2021). FedPacket: A Federated Learning Approach to Mobile Packet Classification. IEEE Transactions on Mobile Computing.

A quick overview of Federated Learning (2)

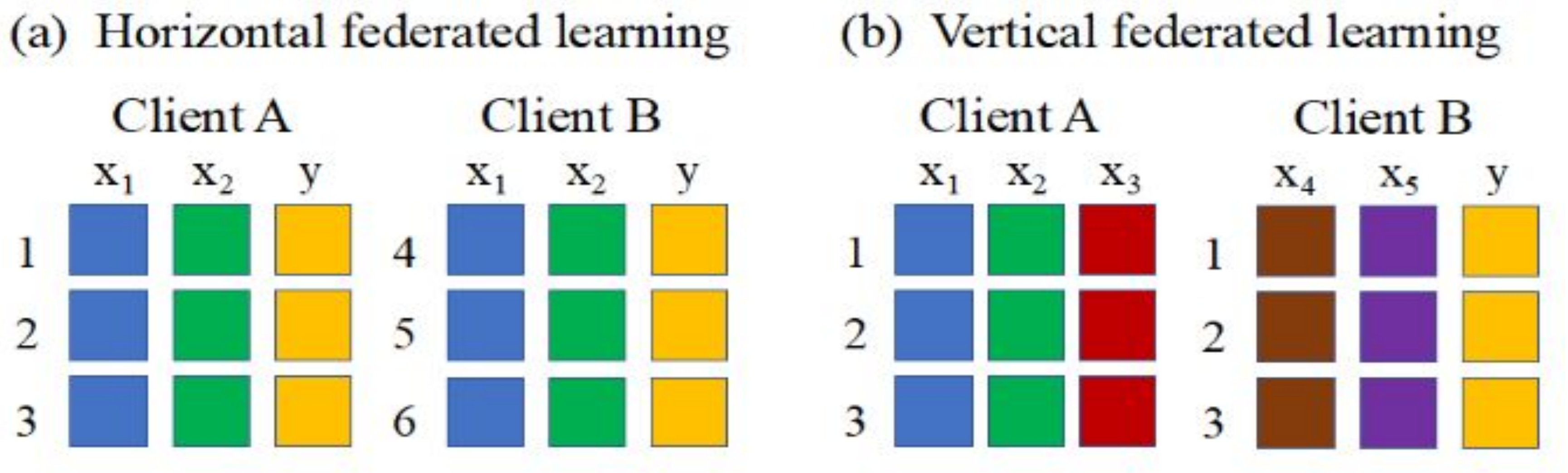
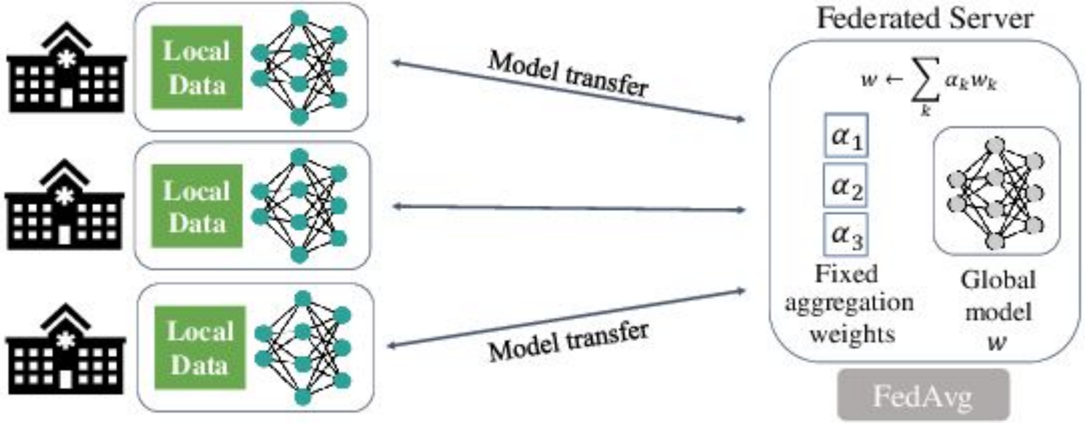


Image extracted from: Chen, Shaoqi & Xue, Dongyu & Chuai, Guohui & Yang, Qiang & Liu, Qi. (2020). FL-QSAR: a federated learning based QSAR prototype for collaborative drug discovery. 10.1101/2020.02.27.950592.

Federated Learning Algorithmic Approaches (1)

- The majority of algorithmic approaches in the domain of FL are derived from the original proposal of Federated Averaging (FedAvg)



Does this approach work for non-NN/DL models?

Image extracted from: Xia, Yingda, Dong Yang, Wenqi Li, Andriy Myronenko, Daguang Xu, Hirofumi Obinata, Hitoshi Mori, Peng An, Stephanie A. Harmon, Evrim B Turkbey, Baris I Turkbey, Bradford J. Wood, F. Patella, Elvira Stellato, Gianpaolo Carrafiello, Anna Maria Ierardi, Alan Loddon Yuille and Holger R. Roth. "Auto-FedAvg: Learnable Federated Averaging for Multi-Institutional Medical Image Segmentation." *ArXiv* abs/2104.10195 (2021): n. pag.

Federated Learning Algorithmic Approaches (2)

- Let's analise fuzzy c-means algorithm

1. Initialize $U = [u_{ij}]$ matrix, $U^{(0)}$
2. At k-step: calculate the centers vectors $C^{(k)}=[c_j]$ with $U^{(k)}$:

$$\mathbf{c_j} = \frac{\sum_{i=1}^{N} u_{ij}^{\lambda} * x_i}{\sum_{i=1}^{N} u_{ij}^{\lambda}}$$

3. Update $U^{(k)}$, $U^{(k+1)}$

$$\mathbf{u_{ij}} = \frac{1}{\sum_{k=1}^{C} \left(\frac{\|\mathbf{x}_i - \mathbf{c}_j\|}{\|\mathbf{x}_i - \mathbf{c}_k\|}\right)^{\frac{2}{\lambda-1}}}$$

4. If $\|U^{(k+1)} - U^{(k)}\| < \epsilon$ then STOP; otherwise return to step 2

where ϵ is a termination criterion between 0 and 1.

What about if we have our dataset distributed in M clients?

$$\mathbf{v_c} = \frac{\sum_{m=1}^{M} \boxed{\sum_{i=1}^{N_m} (u_{ic}^{m})^{\lambda} * x_i^{m}}}{\sum_{m=1}^{M} \boxed{\sum_{i=1}^{N_m} (u_{ic}^{m})^{\lambda}}}$$

All these M clients must send the information in red.

More details in: CORCUERA BÁRCENA, J.L., Marcelloni, F., Renda, A., Bechini, A. and Ducange, P., 2021. A federated fuzzy c-means clustering algorithm. In *CEUR WORKSHOP PROCEEDINGS* (Vol. 3074, pp. 1-9). URL: https://ceur-ws.org/Vol-3074/paper08.pdf

Existing FL frameworks

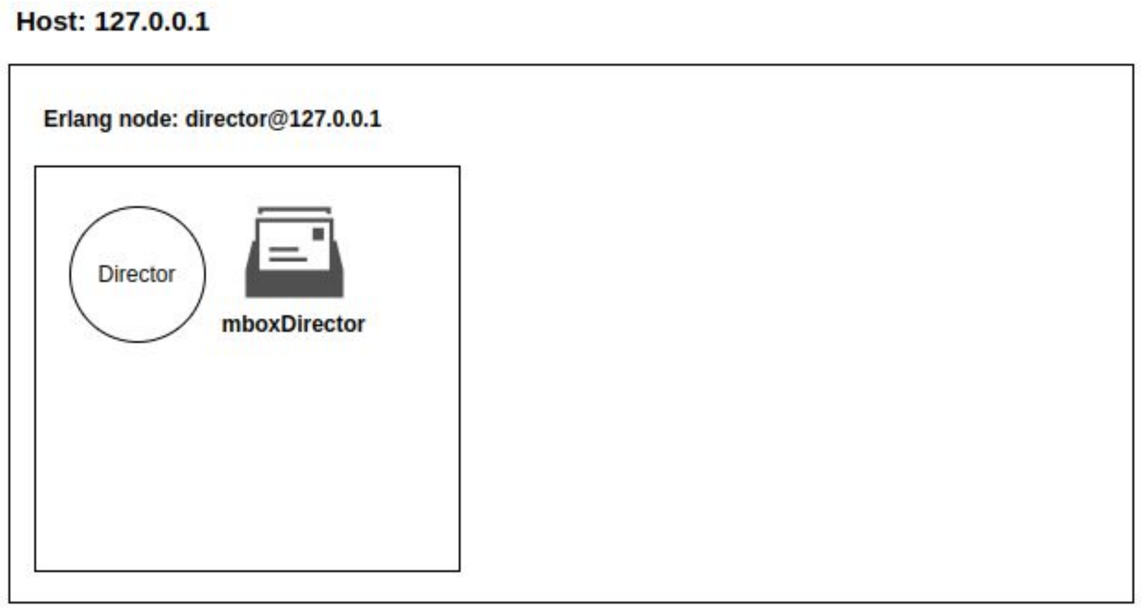
- Purely development in Python
- Communication based on gRPC
- Designed to work with Neural Networks or Deep Learning models
- Round based on two operations: fit and evaluate, or train and predict

FedLang Overview

- Fed-Lang is a middleware designed for supporting the communication between FL participants
- Data Scientists are only in charge of implementing machine learning algorithms in the Python programming language
- It supports transmission of complex data structures
- Participants can be join multiple FL experiments at the same time
- It is still under development 

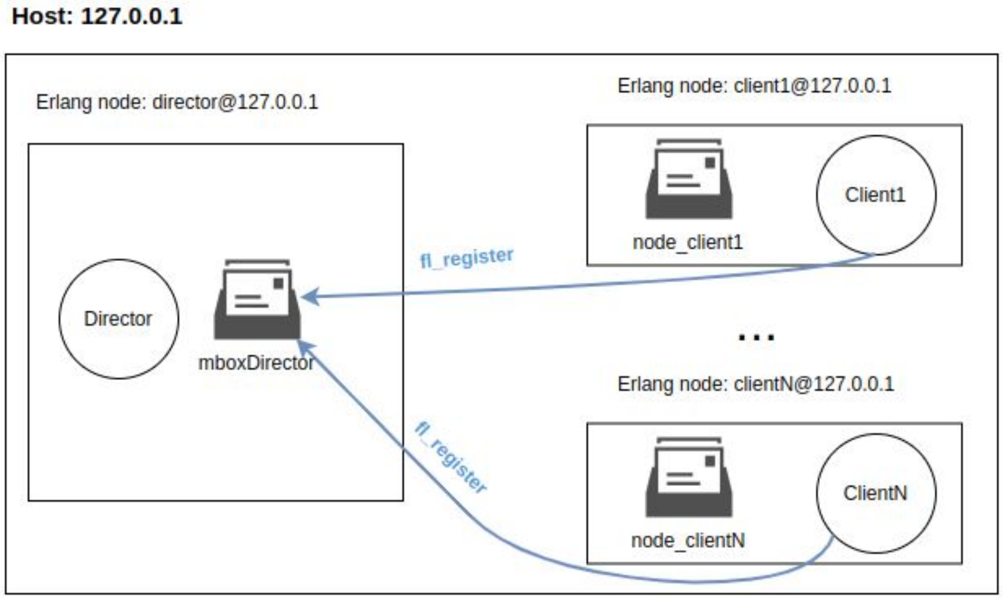
FedLang: Example of FL experiment execution (1)

- The Director is a process in charge of the managing participants and FL experiments



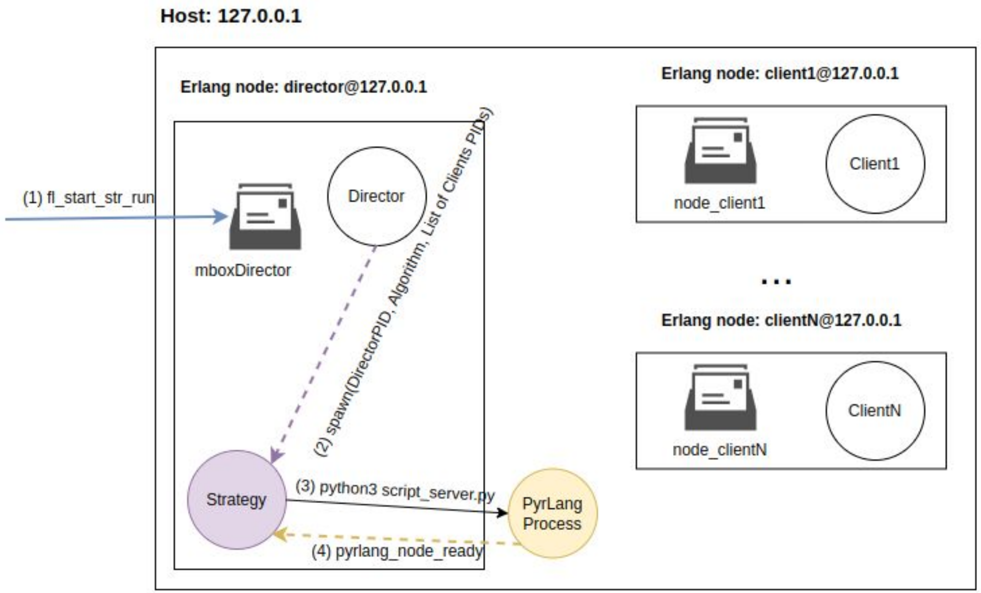
FedLang: Example of FL experiment execution (2)

- A participant/client node sends a message to the Director in order to register itself for participating in FL experiments



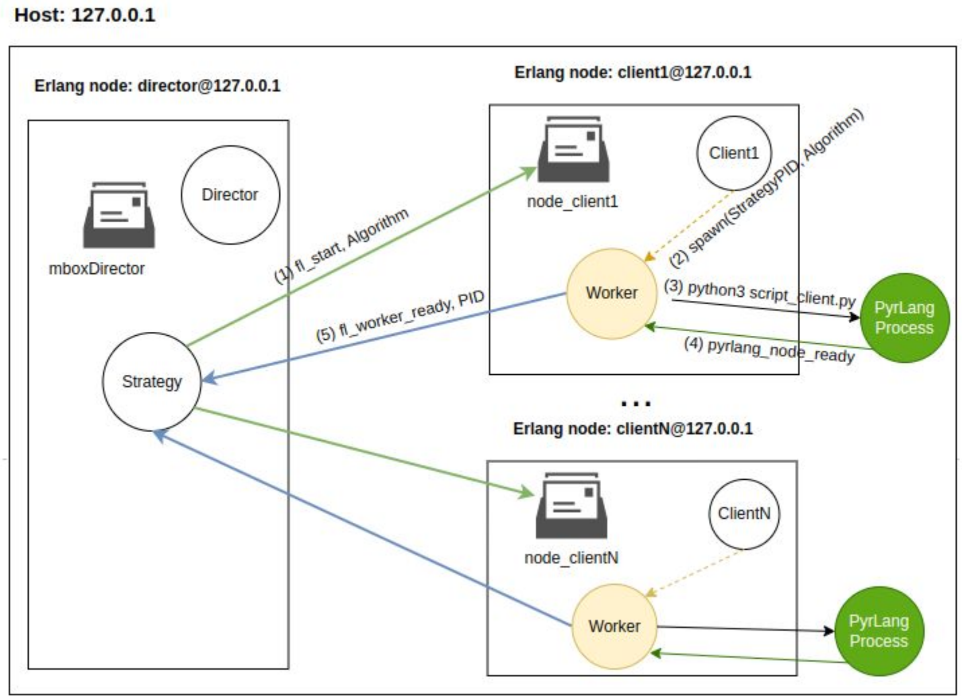
FedLang: Example of FL experiment execution (3)

- Upon the arrival of a message fl_start, the Director will start a new FL experiment. As a result, a Strategy process is created and it will instantiate a Python process



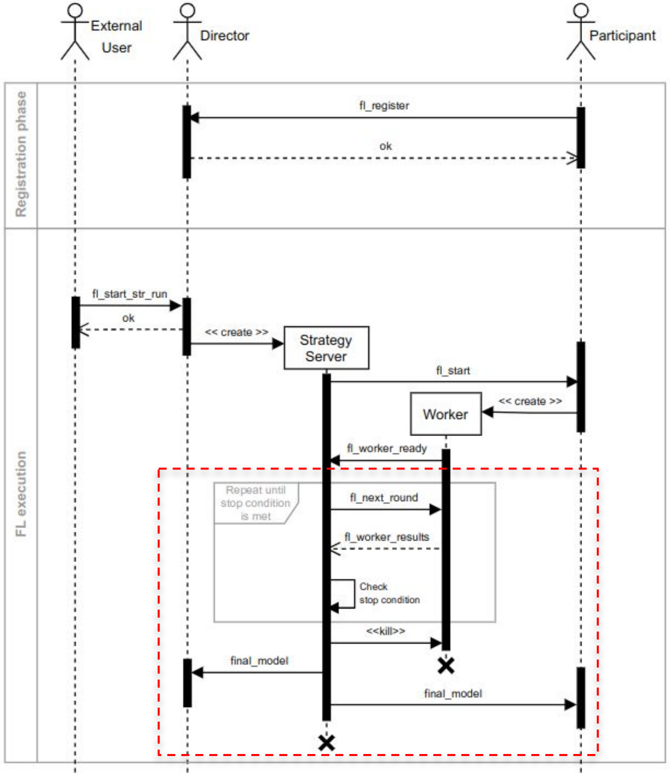
FedLang: Example of FL experiment execution (3)

- The Strategy process notifies participants the start of a new FL experiment.
- In each participant side:
  - A local Worker process is created
  - The local Worker creates a Python process
  - The local Worker notifies the Strategy process it is ready



FedLang: Example of FL experiment execution (4)

- Once Strategy receives all "fl_worker_ready" messages, the federation starts
- A federation consists in an iterative process where some messages are exchanged between Strategy and Workers process until a stop condition is reached (i.e. the accuracy is higher than a threshold)



FedLang demo

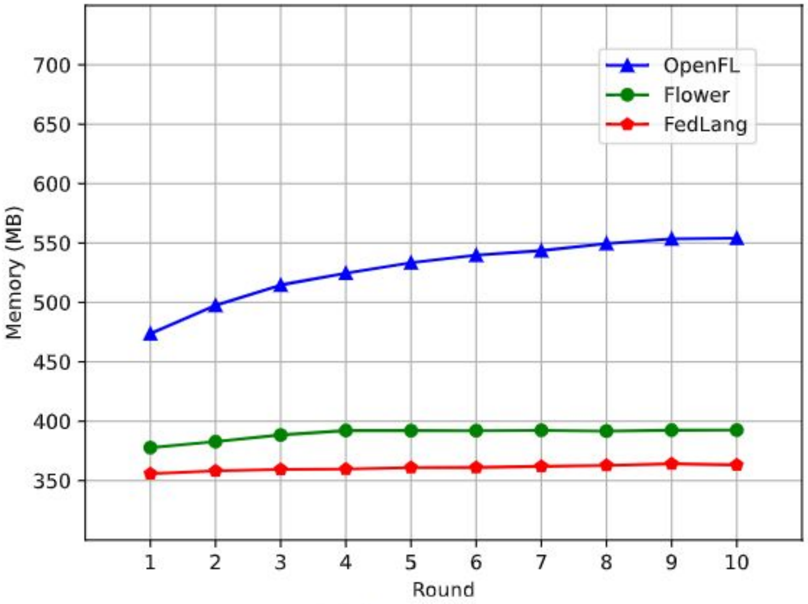
FedLang: Experimental Setup

- Frameworks to evaluate: OpenFL, Flower and FedLang
- Dataset: MNIST, https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
- Number of experiments by framework: 10
- Max number of iterations/rounds per experiment: 10
- Number of clients: 10

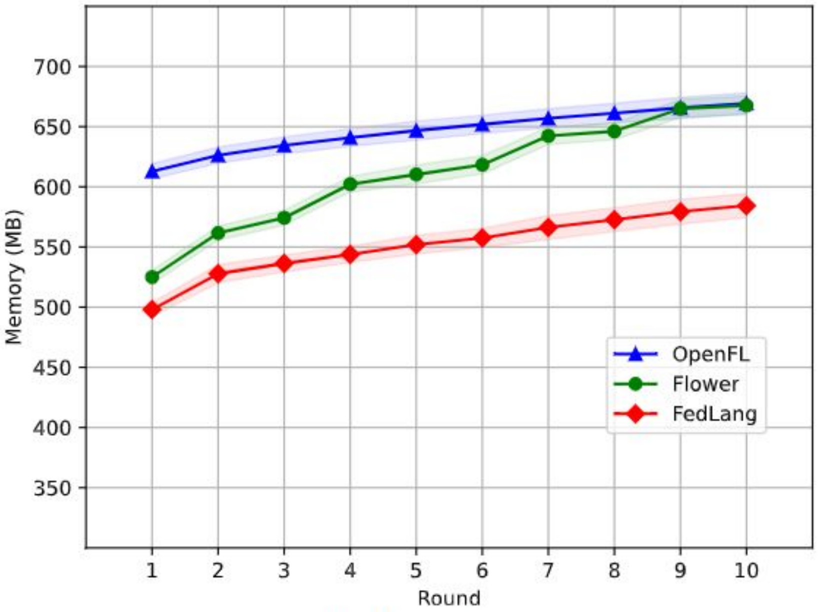
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 13, 13, 16)	272
conv2d_1 (Conv2D)	(None, 5, 5, 32)	8224
flatten (Flatten)	(None, 800)	0
dense (Dense)	(None, 100)	80100
dense_1 (Dense)	(None, 10)	1010

Total params: 89,606
Trainable params: 89,606
Non-trainable params: 0

FedLang: Performance results (1)



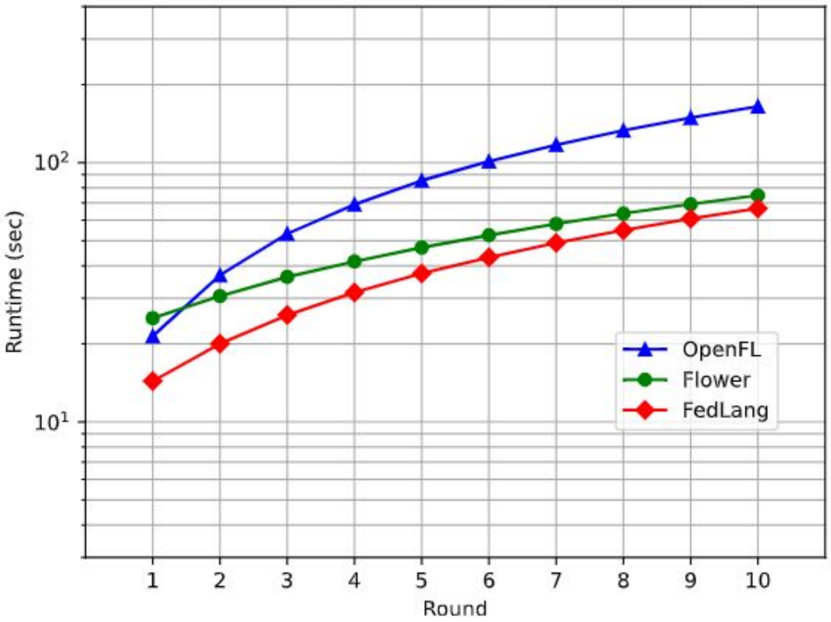
(a) Server-side



(b) client-side

Memory consumption of three state-of-the-art FL frameworks along the execution of the same FL process (implemented in Python), for the coordinator in chart (a), and for data holders in chart (b). Fed-lang shows the lowest memory footprint.

FedLang: Performance results (2)



Timings (in log scale) for the completion of each round of the first experiment on different frameworks.

Pyrlang in Action (1)

- Like Jinterface, Pyrlang is a Python library that enables Python applications interact with Erlang processes
- This library depends on **term** library. This library defines the equivalent Erlang data types but in Python
- Pyrlang defines two main classes:
  - pyrlang.node.Node
  - pyrlang.process.Process

Pyrlang in Action (2)

- To create a Pyrlang application:
  - Make sure you have Python 3.x installed
  - Have cloned Term and Pyrlang
    - https://github.com/Pyrlang/Term
    - https://github.com/Pyrlang/Pyrlang
  - Once you have cloned those repositories, access to each of them and run the following command: pip install -e .
- Now, you are ready!

Pyrlang in Action (3)

- The Node class is used to create an instance of an Erlang Node process

```
from pyrlang.node import Node

node_name = "pyrlang_node@127.0.0.1"
cookie = "abcde"
mailbox = "nodeMailBox"

n = Node(node_name=node_name, cookie=cookie)

n.run()
```

- You can run the command “epmd -names” to see this Pyrlang process

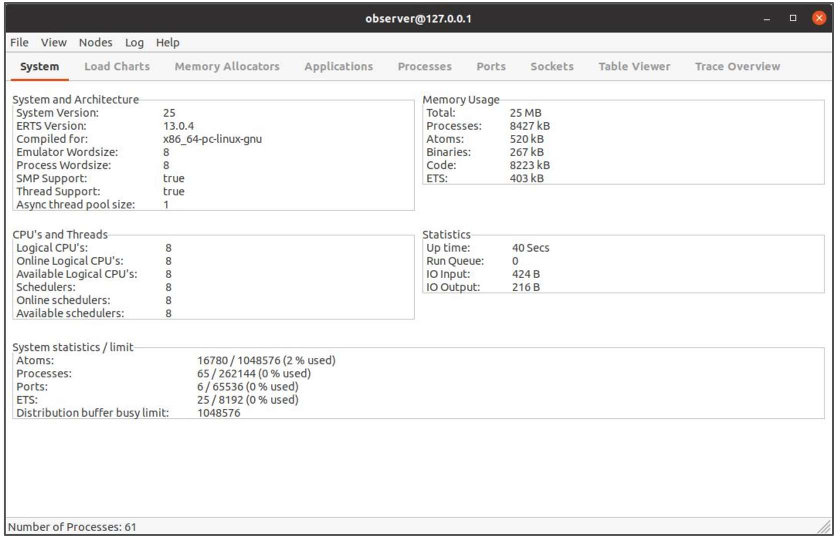
Pyrlang in Action (4)

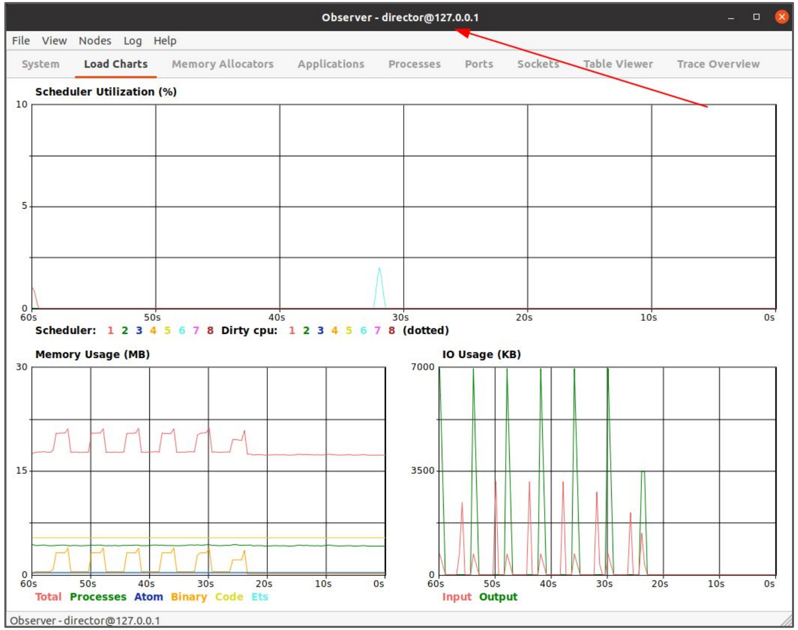
- A Process instance is used to receive incoming messages

```
from pyrlang.node import Node
from pyrlang.process import Process

node_name = "pyrlang_node@127.0.0.1"
cookie = "abcde"
mailbox = "nodeMailBox"

class TestProcess(Process):

    def __init__(self, **kwargs) -> None:
        Process.__init__(self)
        self.get_node().register_name(self, Atom(mailbox))

    def handle_one_inbox_message(self, msg):
        print(f"msg: {msg}")


n = Node(node_name=node_name, cookie=cookie)
TestProcess()
n.run()
```

Monitoring Erlang nodes

erl -name observer@127.0.0.1 -hidden -setcookie "cookie_123456789" -run **observer**





References

- https://github.com/Pyrlang/Term
- https://github.com/Pyrlang/Pyrlang
- http://erlport.org/
- https://www.erlang.org/doc/apps/observer/observer_ug.html
- https://wiki.python.org/moin/PythonSpeed/PerformanceTips