Pervasive AI

ECML-PKDD 2023 Tutorial

Davide Bacciu, Antonio Carta, Patrizio Dazzi, Claudio Gallicchio
University of Pisa, Italy

Solutions and Infrastructures for distributed and federated learning

Davide Bacciu, Antonio Carta, **Patrizio Dazzi**, Claudio Gallicchio
University of Pisa, Italy

http://pai.di.unipi.it/aaai-2023-tutorial-on-pervasive-ai/

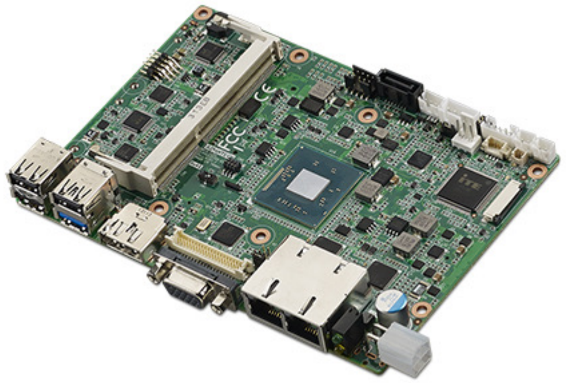
Outline

- Need for going beyond single machine learning
- Distributed learning
- Federated learning
- Beyond federated learning
- How to approach the development of such stuff
- Conclusions

Need for going beyond single machine for learning

One Machine ~~The World~~ is not enough

- ✓ Learning using single machines could be limiting
  - ✓ Computational limitations
    - ✓ a single machine may not have **enough computational power** to train large models in a reasonable amount of time
  - ✓ Memory limitations
    - ✓ large models require **more memory** than a single machine can provide
  - ✓ Scalability challenges
    - ✓ a single machine may not be able to handle the increase **in data size and complexity** when training large models

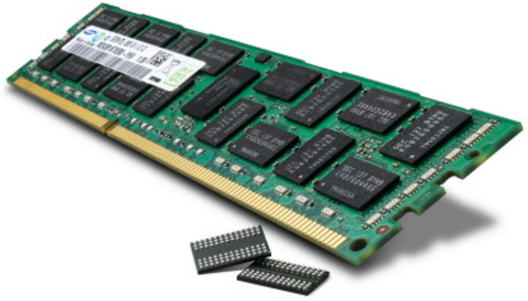


Computational Limitations

A single machine does not have enough computational power to train large models in a reasonable amount of time

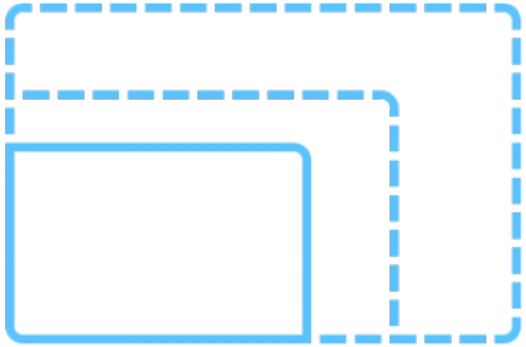
Insufficient computational power can lead to: longer training times, decreased accuracy

Memory Limitations



The **greatest** challenges when training large models using a single machine

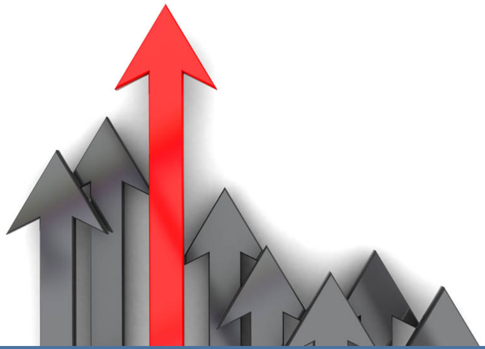
Scalability Challenges



Scalability refers to the **ability** of a **system** to perform well under an **increased or expanding workload**.

A system that scales well will be able to **maintain or increase its level of performance** even as it is tested by **larger and larger** operational **demands**.

a single machine may not be able to handle the **increase** in data size and **complexity** when training large models

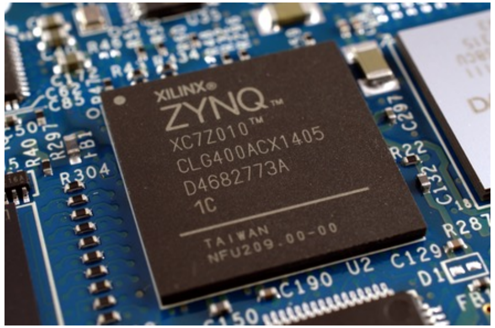


Any way to improve single machine capabilities?

To some extent...

Multicores

GPUs



FPGAs

Parallel Computing	Distributed Computing
Many operations are performed simultaneously	System components are located at different locations

Parallel Computing	Distributed Computing
Many operations are performed simultaneously	System components are located at different locations
Single computer is required	Uses multiple computers

Parallel Computing	Distributed Computing
Many operations are performed simultaneously	System components are located at different locations
Single computer is required	Uses multiple computers
Multiple processors perform multiple operations	Multiple computers perform multiple operations

Parallel Computing	Distributed Computing
Many operations are performed simultaneously	System components are located at different locations
Single computer is required	Uses multiple computers
Multiple processors perform multiple operations	Multiple computers perform multiple operations
It may have shared or distributed memory	It have only distributed memory

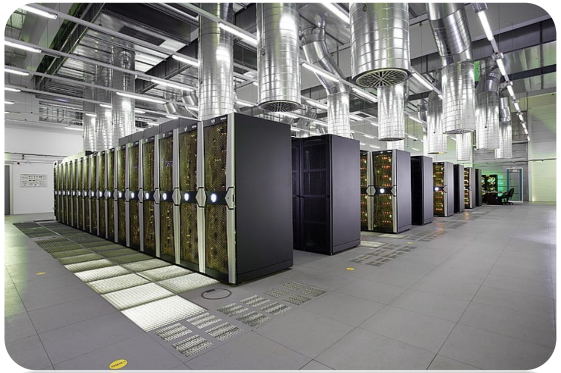
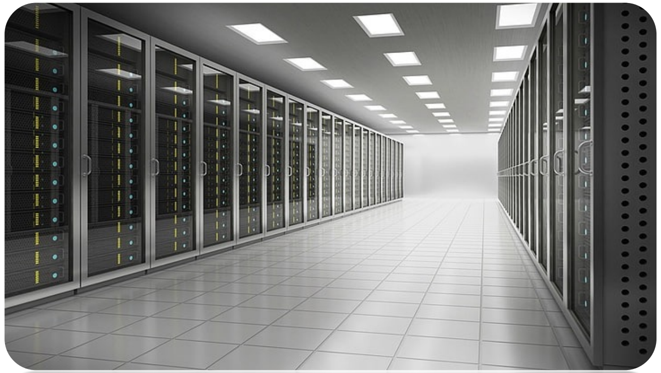
Parallel Computing	Distributed Computing
Many operations are performed simultaneously	System components are located at different locations
Single computer is required	Uses multiple computers
Multiple processors perform multiple operations	Multiple computers perform multiple operations
It may have shared or distributed memory	It have only distributed memory
Processors communicate with each other through bus	Computer communicate with each other through message passing.

Parallel Computing	Distributed Computing
Many operations are performed simultaneously	System components are located at different locations
Single computer is required	Uses multiple computers
Multiple processors perform multiple operations	Multiple computers perform multiple operations
It may have shared or distributed memory	It have only distributed memory
Processors communicate with each other through bus	Computer communicate with each other through message passing.
Improves the system performance	Improves system scalability, fault tolerance and resource sharing capabilities

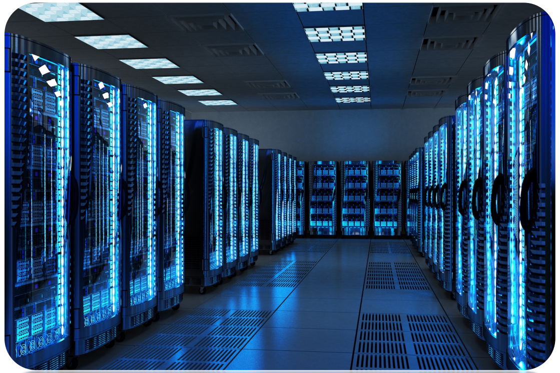
…but the main issue with “not-distributed” parallel machines is on…

SCALABILITY when problem **complexity increases** and **data grows**

This sounds familiar, right?

What to do when a machine is not enough ?



“When the going gets tough, the tough get going.”

one machine is not enough!

Distributed Learning

What is distributed learning ?

Learning performed using a Distributed System!

What is a distributed system?

Various definitions have been given

DEFINITION

- none of them completely **satisfactory**
- none of them in **agreement** with any of the others

"A distributed system is a **collection of autonomous** computing elements that **appears to its users as a single coherent system**."

[Distributed Systems 3, Tanenbaum & Van Steen]

This definition refers to two key features

- ➡ A distributed system is a collection of computing elements, each being able to **behave independently** of the other
- ➡ End users (humans or software) believe they are **dealing with a single system**

This means that one way or another the autonomous nodes need to **collaborate**.

How collaboration happens in distributed systems?

nodes can act independently from each other

➡ nodes need to achieve **common goals** realized by **exchanging messages** with each other

➡ nodes **react to messages** leading to **further communication** through **message passing**

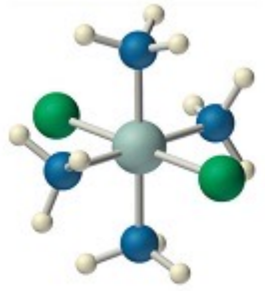
End users should **not even notice** that processes, data, and control are **dispersed across a computer network**

Single coherent system

Coherent if it behaves according to the expectations of its users in a **single coherent system**

The **collection of nodes** as a whole operates the same, no matter where, when, and how **interaction takes place**

This so-called **distribution transparency** is an important design goal of distributed systems.



But let’s avoid to go too far

... let’s just focus on how distributed systems support learning!

How distributed learning is performed?

Namely, how do machines collaborate to speed-up the computation?

It’s a matter of data and computation distribution and synchronisation!

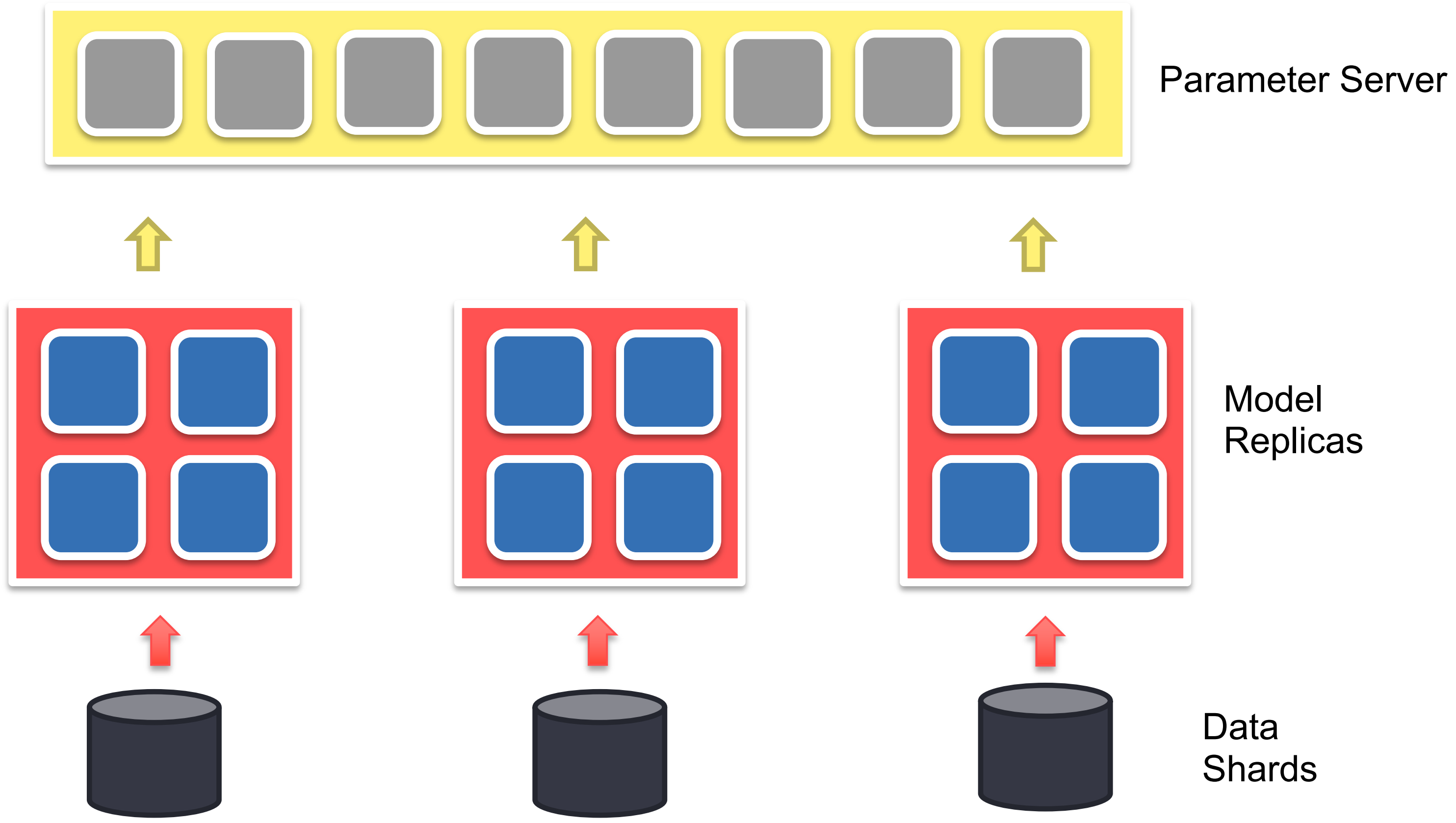
Two flavours of Distributed Training: Data Parallelism vs Model Parallelism

Data Parallelism vs. Model Parallelism

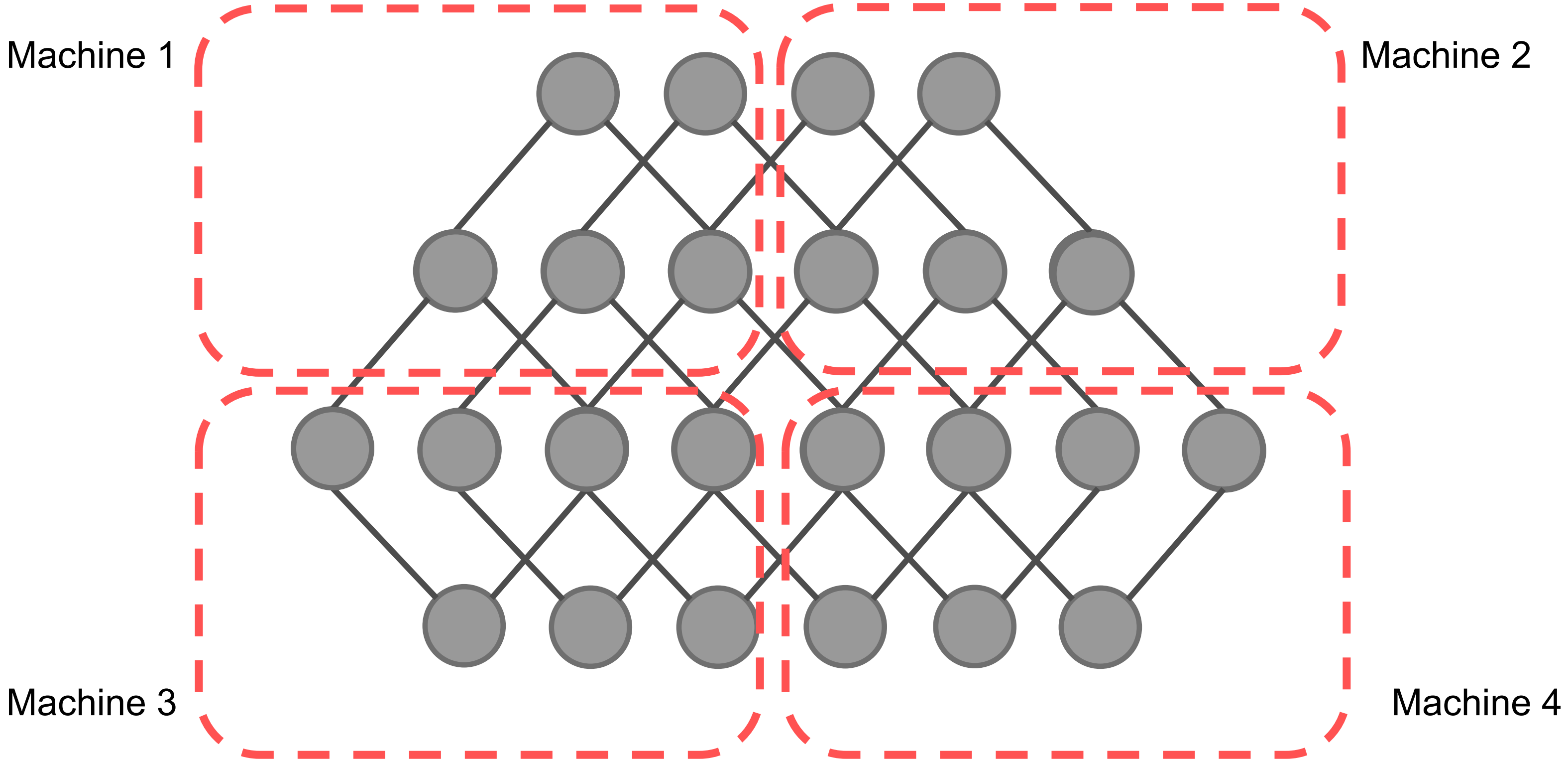
data is **scattered** throughout a **set of machines** that perform the **training loops** in all of them either **synchronously** or **asynchronously**

When a model is so big that it **doesn't fit in the memory of a single device (heard about LLMs ?)**, it is possible to divide it into different parts, distribute them across **multiple machines** and train each one of them **independently** using the same data

Data Parallelism



Model Parallelism



Communication and Synchronization Models

Fundamental to understand how the **nodes synchronise** with each other

Relevant as different workers **may work at different speeds** and hence the **partial gradients** may not be available from all the workers at the same time

During the gradient update phase

Using stale gradients then convergence may be slower

Waiting for all workers to finish, then it may not be very efficient.

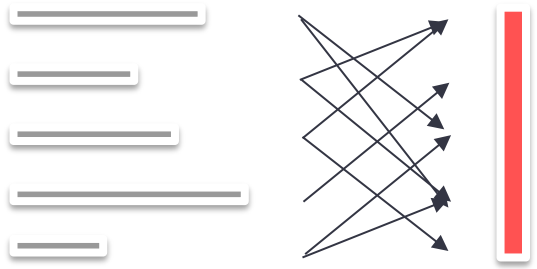
Three models of communication arise as a result of the tradeoff between speed and convergence.

Bulk Synchronous Parallel - BSP

BSP is, at the most basic level, a **two-step** process performed **iteratively** and **synchronously**:

1) perform **task computation on local data**
2) communicate **the results**, and then repeat the two steps.

Thus the BSP model is composed of the **workers**, the **communication between them** and a **barrier**



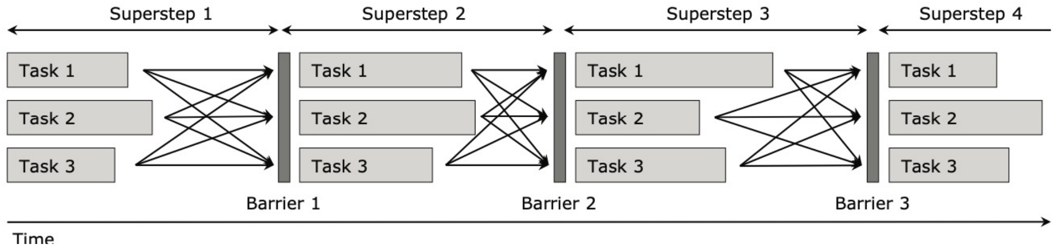
The **barrier** marks the end of a **super step** or an **iteration**

Bulk Synchronous Parallel - BSP

In the Distributed ML case each **worker** works on its own **gradient** and the **barrier** ensures that the **parameter server** updates the weights only when it receives the gradient from **all the workers**

The BSP model trades off speed for convergence.

synchronization of the parallel tasks occur at the super step barriers, depicted below



Asynchronous Parallel - ASP

With ASP, all workers send their gradients to the server, but **no synchronisation is implemented**

Workers **do not wait for other workers to complete**; hence, the parameter server may have **stale gradients** from a few workers.

This causes **errors in the gradient calculation** and hence **delays the convergence**. Also, each worker may obtain **a different version** of the weight from the parameter server.

Consequently, **ASP has the least training time but** yields a **lower accuracy** and is **not stable** in terms of model **convergence**

SYNCHRONOUS
VERSUS
ASYNCHRONOUS

Stale Synchronous Parallel - SSP

SSP **combines ASP and BSP** and uses a policy to switch between ASP and BSP during training dynamically

The idea is that **the difference in the iteration number** for the fastest and the slowest worker **should not exceed** a user-defined number

There is **no waiting time**, but the **fastest workers may have to wait** for the slowest worker to catch up

The model **convergence guarantee** is high but **decreases as the staleness increases**

Existing Frameworks

PyTorch

HOROVOD

deepspeed

RAY

TensorFlow

What is still an issue

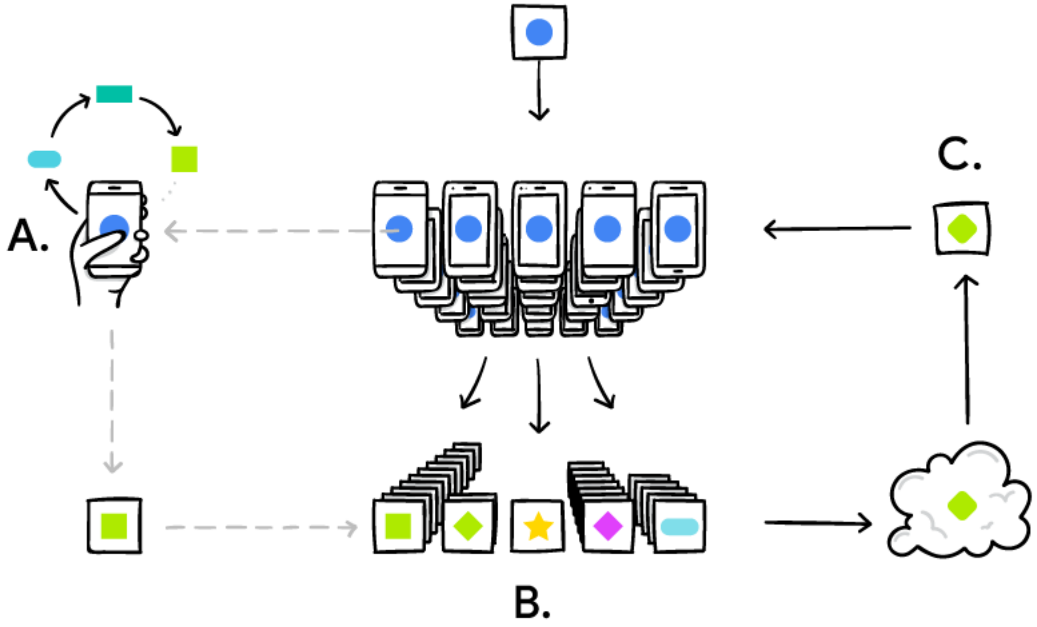
Distributed learning is a way to address the limitation of a single machine

However, Distributed learning, in practice, still needs data to be collected on a cluster or cloud

When users generating data are in order of millions (or even greater) this implies:

- ➡ scalability concerns
- ➡ privacy concerns

Do we have an answer for these issues?



Let’s play with Federated Learning!

Federated Learning

Intro e Motivation

Federated learning **trains a model across multiple decentralised networked devices** holding **local data** samples **without exchanging** them

Federated learning enables **multiple actors** to build a robust machine learning model **without sharing data**, thus addressing critical issues such as:

- data privacy
- data security
- data access rights
- access to heterogeneous data

Federated vs. Distributed (data parallel) Learning

On the assumptions at the basic goals:

- ➡ distributed (data parallel) learning originally aims at **parallelising computations**
- ➡ federated learning aims at **training on heterogeneous datasets**

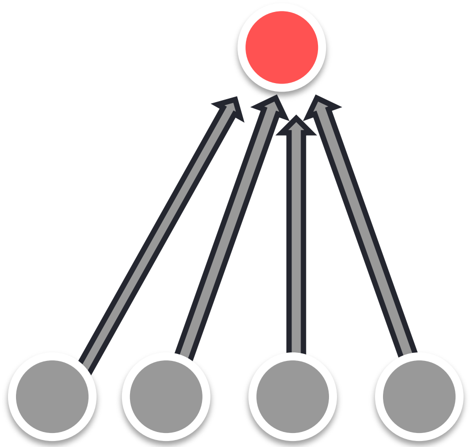
Distributed (data parallel) learning aims at training a single model on multiple servers

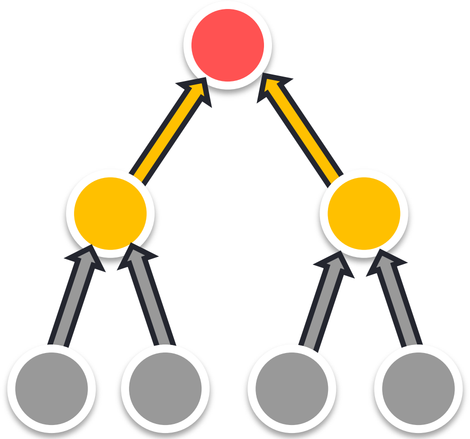
- a common assumption is that the local datasets are **independent and identically distributed**
- roughly have the **same size**

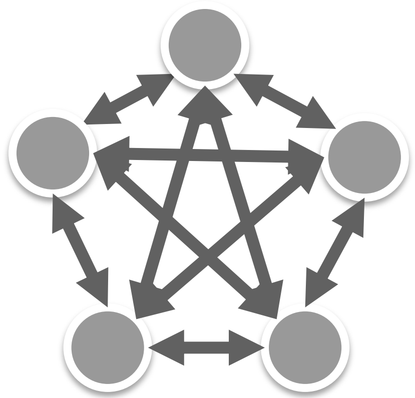
With **Federated learning** the datasets are typically heterogeneous and their sizes vary

Actors involved in federated learning may be unreliable as they are subject to more failures or drop

Aggregation architectures


centralised


hierarchical

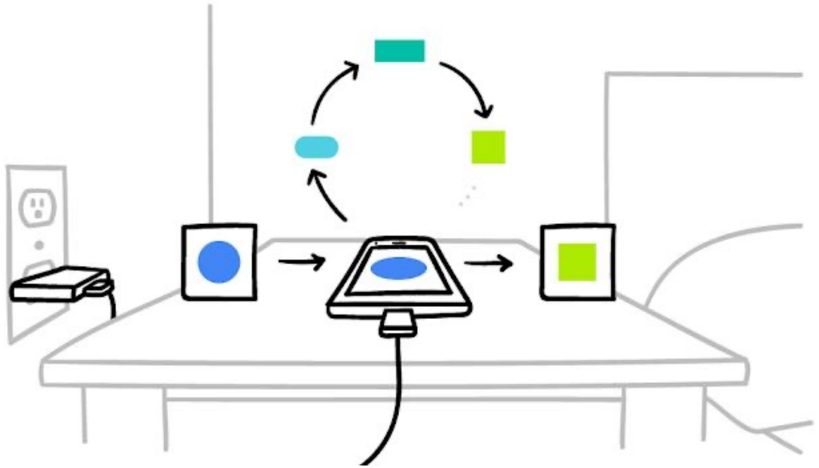

fully decentralised

First-class entities

Actor-centric design and development model

Topology definition

Synchronisation policies



Existing Frameworks

FATE Substra Foundation

Industry-oriented
Scarce prototyping tools

PySyft

Mostly for federated analytics

intel OpenFL

Flower

Only client-server approaches
Lack of flexibility for new methods

Bound to TF logic
Challenging to implement new algorithms

FedRay as a way to perform experimentations in Federated Learning

An R&D-Oriented Framework for easy, end-to-end experimentation in Federated Learning

Rapid prototyping and **evaluation** of FL algorithms via:
- Well-known off-the-shelf algorithms
- API for implementing new algorithms

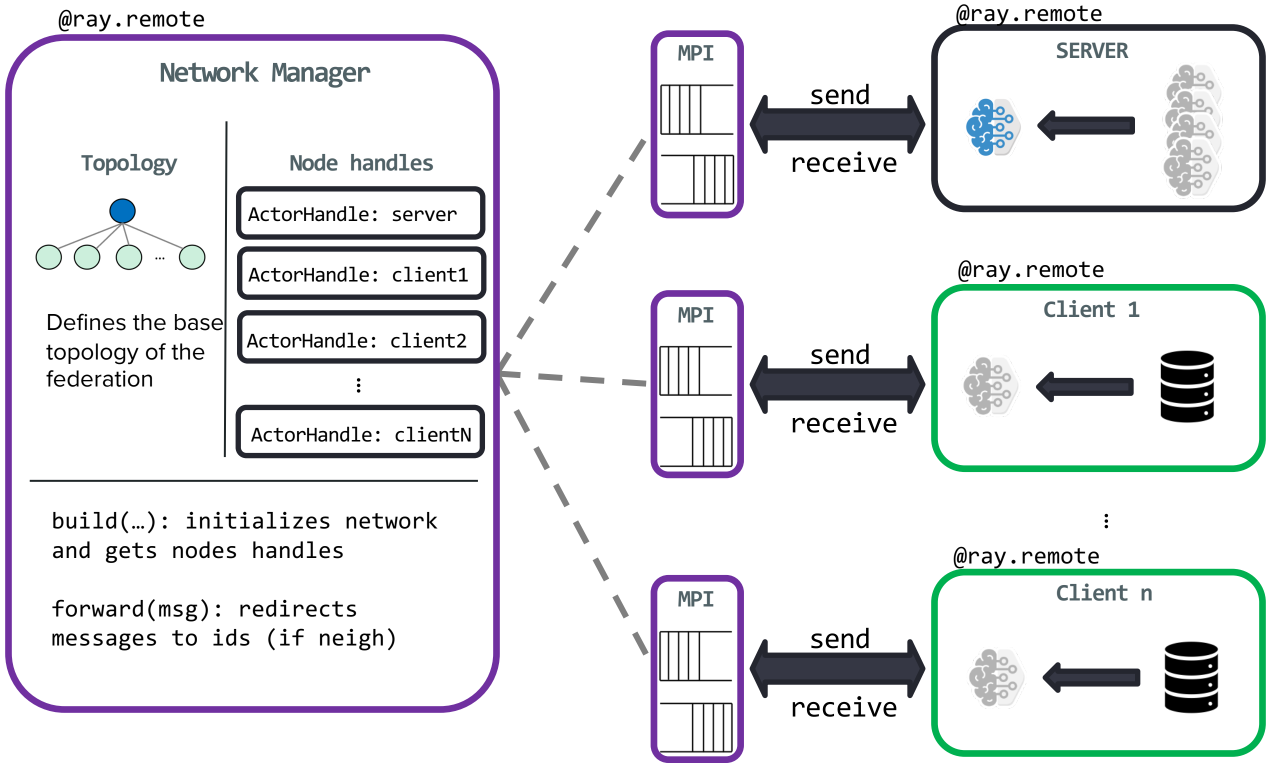
Works with any federation scheme: **Client-Server, Hierarchical, Decentralized**

Both **synchronous** and **asynchronous**

Based on Ray **seamless multiprocessing on any Ray Cluster**

Completely Pythonic API easy implementation and execution

FEDRAY



- Network Manager:
  - Takes care of the network topology (can be dynamic)
  - Keeps references of all the active nodes
  - Forwards messages to participants (only the hex code of the objects in the Ray Object Store)

- Node:
  - Implements the local logic of the federated process
  - Can be either *internal* or *external*
  - Communicates with others via send and receive

FEDRAY

Key use cases

Autnomous driving cars: high number of agents, need to quickly respond to real world situations. Federated learning as a solution for limiting volume of data transfer and accelerating learning

Industry 4.0: privacy of sensitive data for manufacturing companies is of key importance. Federated learning algorithms can be applied to these problems as they do not disclose any sensitive data

e-health: the ability to train machine learning models at scale across **multiple medical institutions** without moving the data is a critical technology

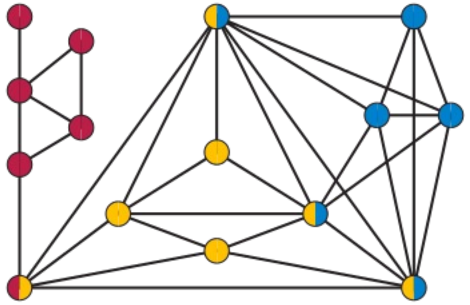
Beyond federated learning

Bringing Decentralised Federated Learning to the next level

Decentralised Federated Learning performances depend on the topology of the network

[H. Kavalionak et al. "Impact of Network Topology on the Convergence of Decentralized Federated Learning Systems," 2021 IEEE Symposium on Computers and Communications, Athens, Greece, 2021]

Adopting “vanilla” approaches for decentralised data exchange (e.g., all-to-all) could lead to inefficient communications



Percolating weights across the network

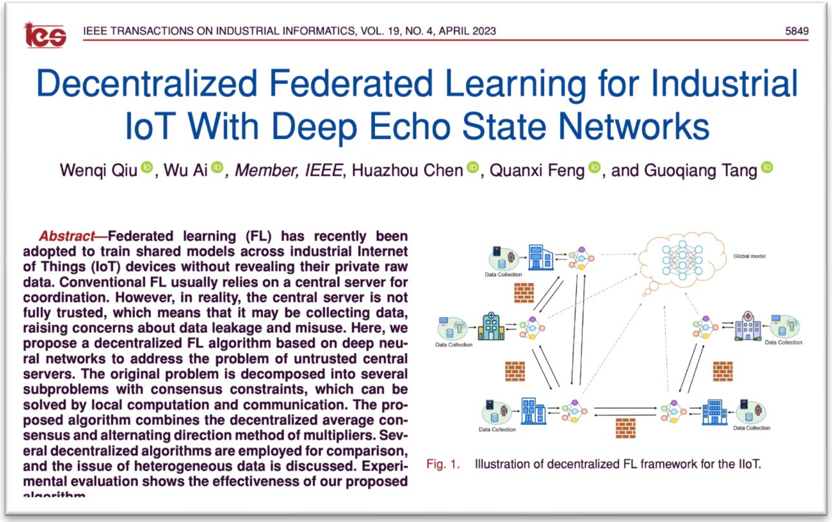
Different approaches have been considered so far to percolate weights across the network in a fully decentralized ways.

Different approaches and different complexities.

Ranging from simple message distribution to overlay networks

Just percolating data across the links...

Some decentralized approaches just send messages across links that computes the average...



IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, VOL. 19, NO. 4, APRIL 2023 5849

Decentralized Federated Learning for Industrial IoT With Deep Echo State Networks

Wenqi Qiu, Wu Ai, *Member, IEEE*, Huazhou Chen, Quanxi Feng, and Guoqiang Tang

***Abstract*—Federated learning (FL) has recently been adopted to train shared models across industrial Internet of Things (IoT) devices without revealing their private raw data. Conventional FL usually relies on a central server for coordination. However, in reality, the central server is not fully trusted, which means that it may be collecting data, raising concerns about data leakage and misuse. Here, we propose a decentralized FL algorithm based on deep neural networks to address the problem of untrusted central servers. The original problem is decomposed into several subproblems with consensus constraints, which can be solved by local computation and communication. The proposed algorithm combines the decentralized average consensus and alternating direction method of multipliers. Several decentralized algorithms are employed for comparison, and the issue of heterogeneous data is discussed. Experimental evaluation shows the effectiveness of our proposed**

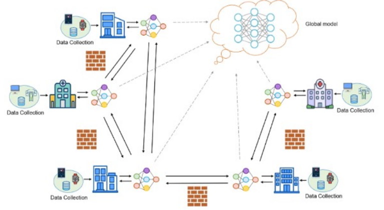
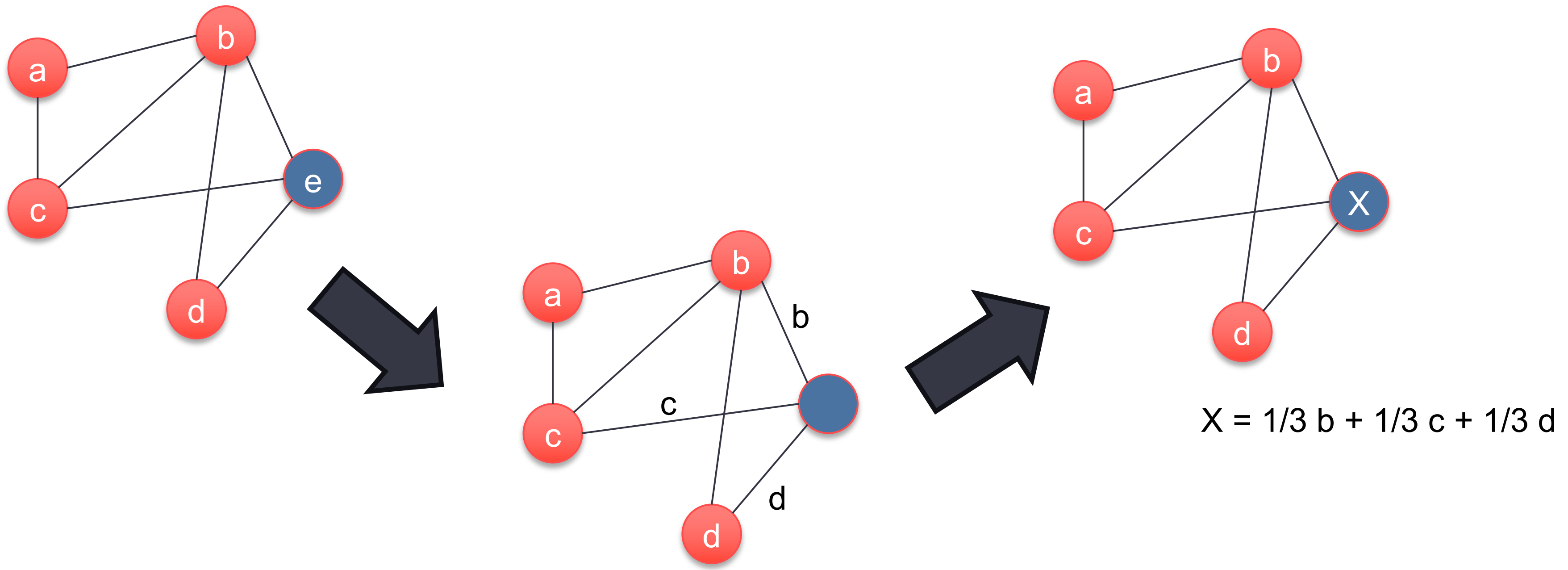


Fig. 1. Illustration of decentralized FL framework for the IIoT.

Just percolating data across the links...



... and computing the average

Gossip Learning

Gossip Learning is a method for learning models from **fully distributed** data **without central control**

[István Hegedűs et al. Decentralized learning works: An empirical comparison of gossip learning and federated learning, Journal of Parallel and Distributed Computing, Vol. 148, 2021]

Each node in the network initialises a **local model $\mathbf{w_k}$** (and its age t_k)

The model is then **periodically sent** to another node in the network **without any synchronisation**

A so-called **sampling service** supports the node selection

Gossip Learning

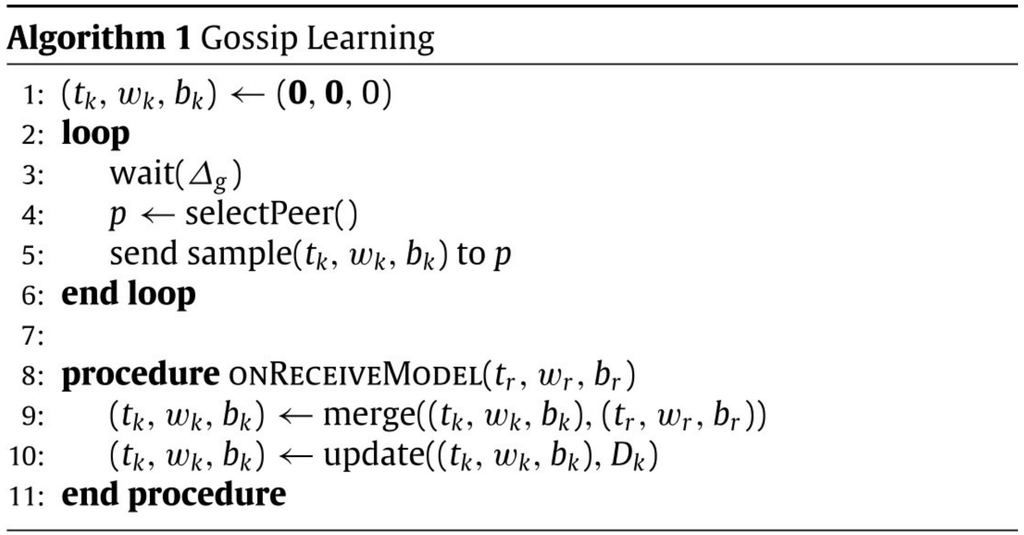
Upon receiving a model w_r, the node **merges** it with the **local model** and updates it using the **local data set D_k**

Merging is achieved **by averaging the model parameters**

In the simplest case, the received model merely **overwrites** the local model

This mechanism results in the models **taking random walks** in the network and being updated when visiting a node

Gossip Learning



Algorithm 1 Gossip Learning

```
$(t_k, w_k, b_k) \leftarrow (\mathbf{0}, \mathbf{0}, 0)$
loop
    wait($\Delta_g$)
    $p \leftarrow$ selectPeer()
    send sample$(t_k, w_k, b_k)$ to $p$
end loop

procedure onReceiveModel$(t_r, w_r, b_r)$
    $(t_k, w_k, b_k) \leftarrow$ merge$((t_k, w_k, b_k), (t_r, w_r, b_r))$
    $(t_k, w_k, b_k) \leftarrow$ update$((t_k, w_k, b_k), D_k)$
end procedure
```

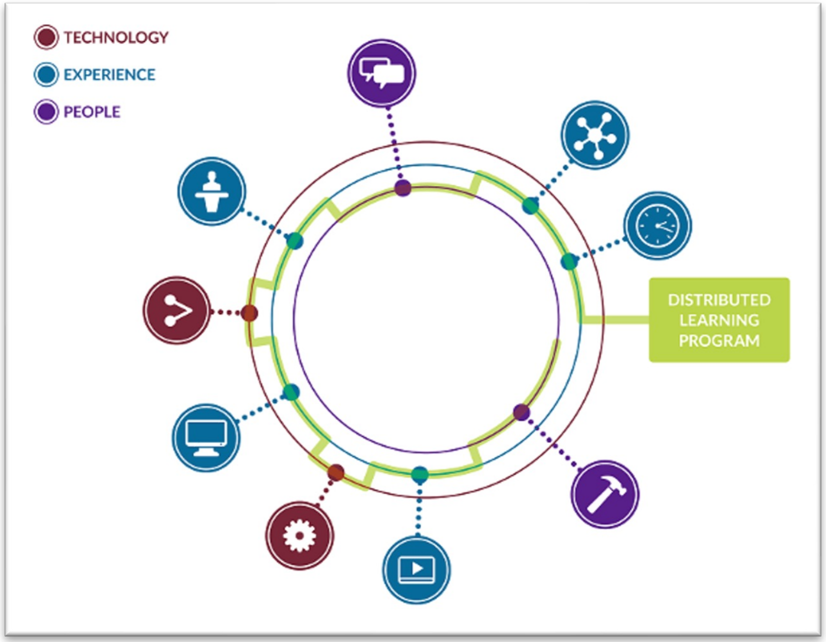
Nice, interesting… but how I am supposed to develop distributed learning applications ?

Writing Applications targeting Distributed and Federated Learning Environments

Different ways to distribute workload

Different mechanisms available

Different assumptions can be made



Programming for Distributed and Federated Learning Environments is complex

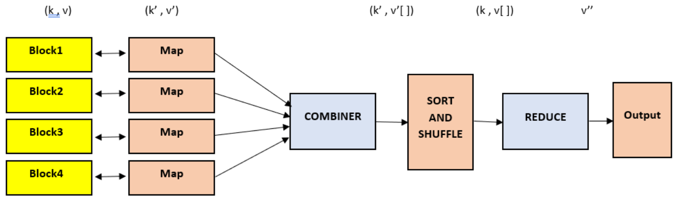
Complexity of orchestrating **multiple devices** and nodes

Need for **programming models** in distributed and federated learning environments

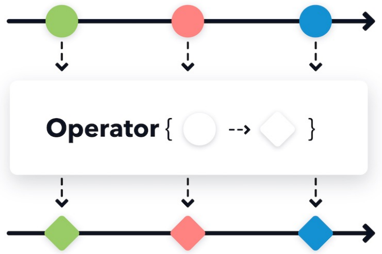
the right programming model can **streamline development and management**

Programming Models for Distributed and Federated Learning Environments

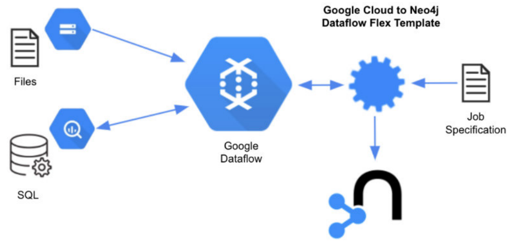
various programming models can be used to target distributed and federated learning environments



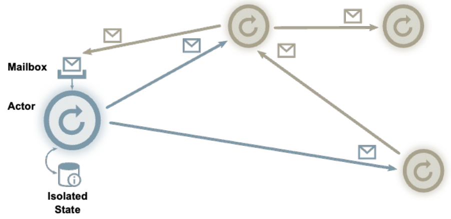
MapReduce



Reactive Programming



Dataflow Programming



Actor Model

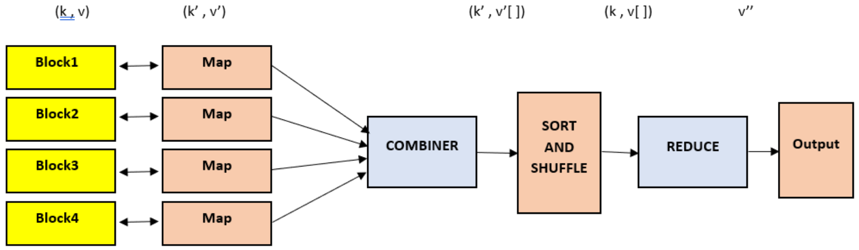
MapReduce

Parallel programming model and processing paradigm used for distributed data processing

Computations as set of iterations involving Map + Reduce operations, mostly not pure functions

Hadoop: the first framework that was available to perform MapReduce computations. Quite low-level approach to the interactions across nodes.
[JVM]

Apache Spark: a framework that improves Hadoop in several ways. A functional-based approach strongly focused on Immutable distributed data structures (RDD)
[JVM]



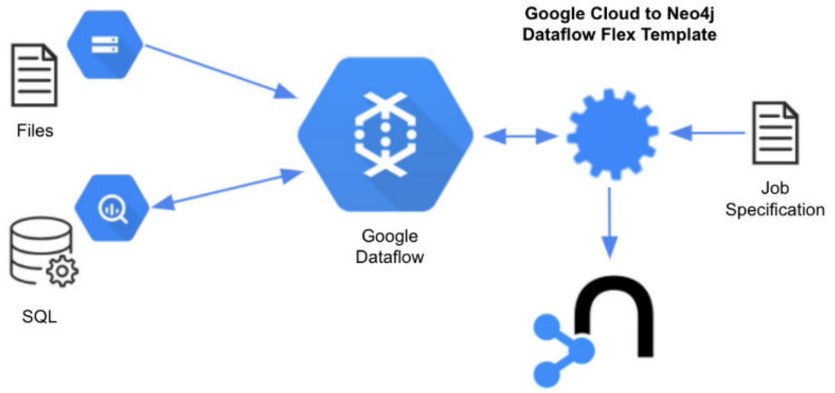
Dataflow Programming

Dataflow Programming as a model for specifying and executing computations as a directed graph of data dependencies

Dataflow Programming is particularly suitable for managing data processing pipelines in distributed systems

Apache Beam: An open-source, unified model for defining both batch and streaming data-parallel processing pipelines.

Google Cloud Dataflow: A fully managed stream and batch data processing service based on Apache Beam, designed for easy deployment and scaling.



Reactive Programming

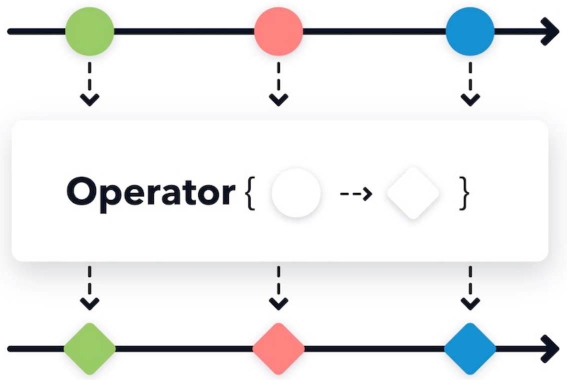
A programming paradigm for **event-driven programming paradigm**

Focuses on **reacting to changes** and **events** in real-time

Reactive Programming can help manage **data streams** and **event-driven systems** in distributed learning environments

RxJava/RxJS: Reactive Extensions for Java and JavaScript, providing a set of libraries for composing asynchronous and event-based programs. [JVM+JS]

Reaqtor is a framework for creating reliable, stateful, distributed, and scalable event processing based on Reactive Extensions (Rx). [.NET]



The Actor Model

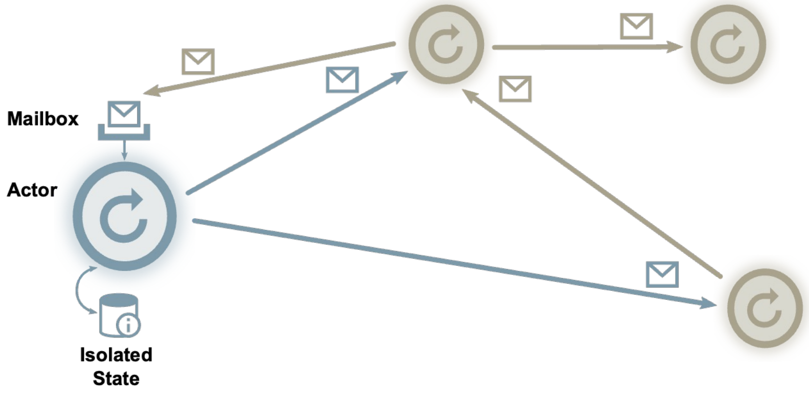
A programming paradigm for **concurrent** and **distributed** systems

Actors are **independent entities** with their **own state** and **behavior**

Actor Model simplifies **handling asynchronous communication** in distributed systems

Akka/Pekko: A popular toolkit and runtime for building highly concurrent, distributed, and fault-tolerant systems. **[JVM]**

Microsoft Orleans: An actor framework designed to simplify distributed application development in .NET, focusing on scalability and reliability. **[.NET]**



Conclusions

Conclusion

When data and models became too big, their training on a single machine is unfeasible

Using more machines is possible, adopting different strategies for distributed learning

When privacy is a concern, federated learning is a possibility

Federated Learning is a vivid research area, with also some nice proposals for going beyond it

Programming (efficiently) with ease distributed learning solutions can be challenging, some approaches exist