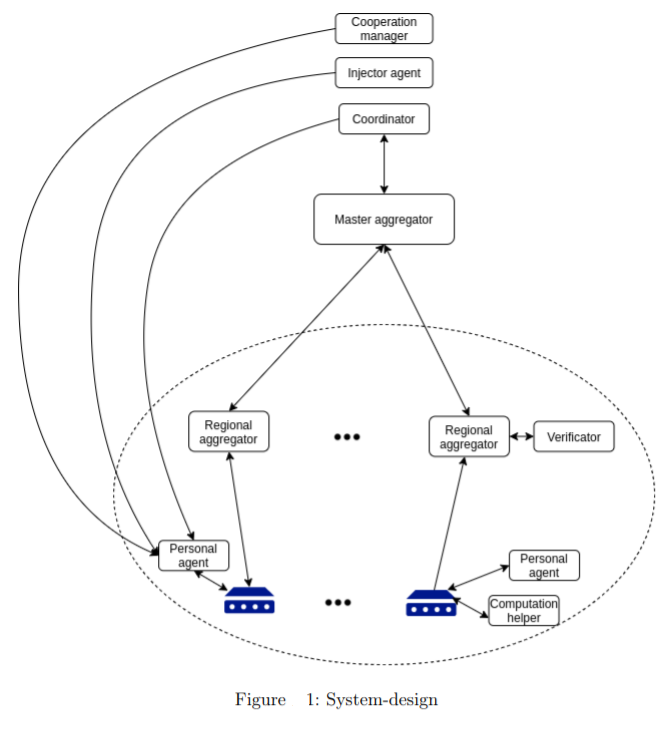
Sunday-FL Inventory

This platform is created using the following technologies: -

* 1. **JAVA**
  2. **AKKA**
  3. **Python (3.8 version)**

**Docker** is used in the project as well. Out of the box docker file for Server is provided. However, the user can create custom docker files for server and clients and run the experiment in a containerized environment.

**AKKA** and **JAVA** are used to implement the below features of the platform: -



* 1. **Injector**

It allows clients to customize their programs and handles a collection of modules required for machine learning. A client would get the module they need to conduct a machine learning assignment.

* 1. **Coordinator**

It's the major player in charge of synchronization and round management. Coordinator is responsible for the whole training process inside a single machine learning activity. To handle each round, the coordinator summons the Master Aggregator.

* 1. **Aggregator**
     1. **Master aggregator**

It keeps track of each machine learning task's rounds. They make dynamic decisions to spawn one or more Regional Aggregators, to whom work is outsourced, in order to grow with the number of devices.

* + 1. **Regional aggregator**

It's a one-round actor. It is in charge of communicating with the Personal Agent. It also notifies the Master aggregator of any changes.

* 1. **Personal agent**

Each device is represented by an actor. It takes part in future agreements with other actors about when to begin learning. It is also in charge of processing learning modules and, if necessary, downloading them. It also invites Selector to participate in the machine learning round.

## Modules to be developed in future are as follows: -

* 1. **Cooperation manager**

It is in charge of setting the specifics of the time slot. Each Personal agent gives a time window in which it could be able to engage in the learning process. The Cooperation Manager determines the appropriate time window based on this. It's vital to note that the above diagram doesn't stipulate how negotiations should take place. It only considers the module's requirement.

* 1. **Verificator**

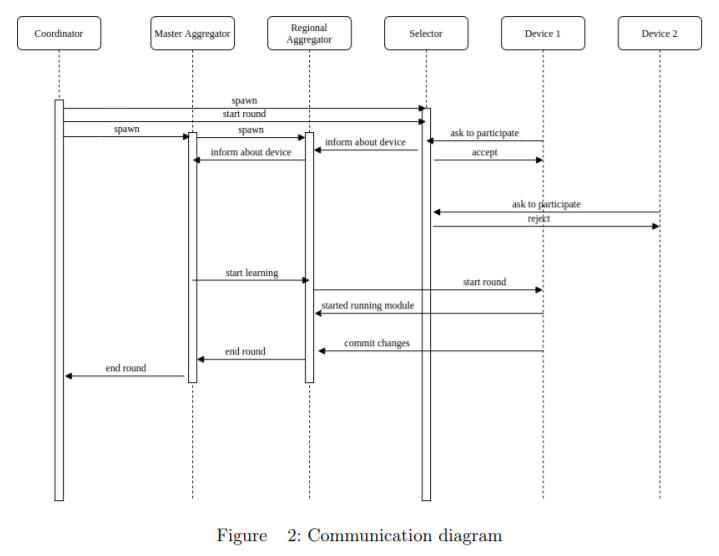
The FL procedure will be verified using this optional actor in the future. In rare circumstances, the client may be acting maliciously and causing harm to the model. In this case, Verificator examines if a model change may enhance it. If this is not the case, the modifications will be refused.

* 1. **Computation helper**

It is intended to make the implementation of complicated tasks easier (in the future). For example, when a gadget, such as an IoT device, is unable to conduct calculations on its own.

# Process flow

* Every device is aware of the server and is aware of its address.
* The server is unaware of the device until it submits a request to participate in learning. Devices initially inform the server of their desire to engage in training.
* For a set length of time, the server collects data from such devices.
* The server then arranges for the training details to be shared with the devices.
* This involves establishing when and on what principles the training will take place, as well as the training model and its different parameters (e.g., neural network).
* The learning step begins after this knowledge has been established.
* Each device must download the necessary modules in order to do training on their local data.



1. The server is up and running and ready to receive devices.
2. Long-lived actors, those who live for more than one round — in this example, they live throughout the entire learning process, such as Coordinators and Selectors, are the first to be initialized.
3. The coordinator actor then sends a message to Selector, instructing it to begin admitting devices into the learning cycle.
4. The Master Aggregator and Regional Aggregator actors are then created as short-lived actors that only last one round.
5. Selectors who have been informed about the commencement of the round are ready to accept devices to participate in the learning process.
6. Selector should get a message from each device indicating their desire to engage in learning.
7. The communication includes a task identification number that corresponds to an issue in which the device wants to take part.
8. Each Selector notifies the Regional Aggregator about the new device, who then notifies the Master Aggregator.
9. The Selector has the option of accepting or declining the request. Because the server currently lacks a policy, it accepts all devices.
10. The server continues to accept devices until the required number has been reached.
11. The server will wait until the required number of devices have registered for the training. If the correct number of devices is not reported for a long enough amount of time, the server will lower the number so that training may commence.
12. When enough devices have joined, the Master Aggregator sends a message to the Regional Aggregator to begin learning, and each Regional Aggregator subsequently sends a message to the device to begin learning.
13. After receiving this communication, the gadget initiates the learning module and informs Regional Aggregator.
14. The current model is transmitted to the training device by the Regional Aggregator. After then, each device uses its own data to train the model.
15. The server updates the model after enough devices have sent in their modifications.

This is how a round ends. The server repeats these rounds until the requisite precision is met.

## Learning Modules

The learning modules are written exclusively in Python 3.8 and Python libraries. There are two core scripts, called as server.py and client.py. The names of the scripts are self-explanatory. Depending upon the learning task, an additional script is created which trains and tests the data and learning model on clients. In the current project it is named as mnist.py. These scripts are highly customizable and can be customized by the user as per the learning requirements.

### Libraries called in the scripts are as follows: -

1. **Torch 1.4.0**

PyTorch is an open-source machine learning library for creating and training deep learning models based on neural networks. It employs dynamic computation, allowing for greater flexibility in the construction of complicated architectures.

1. **Torchvision 0.5.0**

Popular datasets, model architectures, and typical image transformations for computer vision are all included in this library.

1. **Syft 0.2.5**

Deep Learning library in Python that is safe and private. Using Multi-Party Computation (MPC) within PyTorch, PySyft decouples private data from model training.

1. **Logging**

The logging module in Python allows you to write status messages to a file or any other output stream. The file provides some information on which parts of the code are run and what issues have emerged.

1. **Argparse**

It used to include command-line argument parsing. Argparse is used to offer flexibility and reusability to the code by allowing user input values to be parsed and utilized instead of manually setting variables inside the code.

1. **Sys**

Python's sys module has a number of methods and variables for manipulating various aspects of the Python runtime environment. It helps to work with the interpreter since it gives us access to variables and functions that have a lot of interaction with it.

1. **Asyncio**

It's a library for writing concurrent programming that uses the async/await syntax. Asyncio aids in the creation of high-performance networks and distributed task queues, among other things.

1. **numpy**

It's a Python library for manipulating arrays. It also provides functions for working with matrices, Fourier transforms, and linear algebra.

1. **random**

It's a Python built-in module for generating random numbers.

1. **pathlib**

The Pathlib module in Python contains several classes that describe file system paths and have semantics that are acceptable for various operating systems.

1. **json**

The JSON JavaScript Object Notation format is a data structure. Its primary function is to store and send data between the browser and the server. With a built-in module named json, Python also supports JSON.

1. **shaloop**

C optimization to run SHA256 and SHA512 over numpy arrays.

1. **PIL**

Python Imaging Library is a free and open-source extension library for the Python programming language that allows you to access, manipulate, and save a variety of image file types.

1. **os**

Python's OS module includes methods for creating and deleting directories (folders), retrieving their contents, altering, and identifying the current directory, and more.

1. **copy**

The Copy Module is a collection of functions for copying various items of a list, objects, and arrays, among other things. Both shallow and deep copies may be made using it.

## Dataset Module

Configuration of the process can be set up with the help of command line parameters or with the help of json file called partition\_config.json. Available parameters can be divided into three main parts: input data, partitions, output data.

### Input data parameters describe the source dataset:

* + - **datapath** - folder with the input data, example: "../data"
    - **data\_file\_name** - source data filename or directory where the folders with images are located, example: "plants"
    - **target\_file\_name** - if the target variable is stored in a separate file, then its name must be specified here
    - **delim** - if the data is stored in csv format, here the delimeter of the file is specified
    - **target\_name** - name of the target variable inside the dataset if applicable
    - **data\_name** - if the data is stored as an element of dictionary, specify data key here, example: "images" will look for loaded\_data['images']
    - **data\_root** - if the data is stored inside some nested structure, here the path to data is specified, example: node1, node2, node3.

### Partition description:

* + - **partitions\_num** - number of partitions to generate, example: 3 will generate 3 separate data files and 3 target files
    - **partitions\_size** - number of elements inside one partition
    - shuffle - if the data should be shuffled before partitioning
    - **partition\_mode** - three partition modes are available: random, equal, and custom. Random assigns data to partitions randomly, equal tries to assign equal amount of each class per every partition, custom provides a dictionary that describes how many percent of each partition should be assigned to every class (data\_assign parameter).
    - **data\_assign** - explains how data should be organized inside every partition in percentage of total partition. Example of custom partition into 3 pieces:"[{'0': [0.5, 0.5, 0, 0, 0, 0, 0, 0, 0, 0], '1': [0, 0, 0.5, 0.5, 0, 0, 0, 0, 0, 0], '2': [0, 0, 0, 0, 0.5, 0.5, 0, 0, 0, 0]}]", where 0,1,2 - future partitions, [0.5, 0.5, 0, 0, 0, 0, 0, 0, 0, 0] - tells to give one half of partition data to the first class and the other half to the second class

### Output description:

* **save\_to\_directory** - directory, where the resulting files will be saved, example: "../data",
* **data\_prefix** - string that will be used as prefix for saved file with data, number of the partition will be added as '\_ (partition num)' automatically, example: "data\_custom" will produce files like data\_custom\_0.npy, data\_custom\_1.npy and so on
* **target\_prefix** - same as for data but for target files, example: "target\_custom"

# How it works



In the above image, instance refers to respective python files as mentioned in previous section. To summarize the above, for executing the project the user needs to: -

* 1. Prepare dataset via dataset module by executing python scripts.
  2. Moving the training and testing datasets to the requisite client folder.
  3. Updating the config files with the URL and Port on which Server module will be executed and bound to later on.
  4. Update the minimum number of clients, type of learning and other parameters required in the config file.
  5. Start the Server module.
  6. Instantiate the Client modules, with unique names, valid dataset ids and available port numbers with which it could be bound to.
  7. Sometimes the clients fail to start due to windows WebSocket error, just instantiate them again and they will work.
  8. View results once the training is completed.

# Networking – interconnectivity between modules and actors

When the JAVA project is started and the JAVA objects are created, for both server and client and other modules, networking between them is handled via AKKA.

Once the learning is started the networking between SERVER and CLIENT python instances is handled via PySyft. AKKA does not play any part once the chain of command is switched over to python part of the project.

# How is it known that everything is ready, and project is successfully executed?

During each successful calls in Figure 2, a success message is logged into the console window. If there are no errors, and the epochs and accuracy is visible on console, then the project ran successfully.

Text

Description automatically generated

In case of any errors, whether connectivity or library based etc., needs to be debugged manually. The user will see errors for the same logged into the console window. There is a new error in windows where WebSocket’s give an error on initializing clients. The user needs to re-initialize the client once more. This is the current working solution for the particular issue as it is dependent on Vendor (Microsoft).