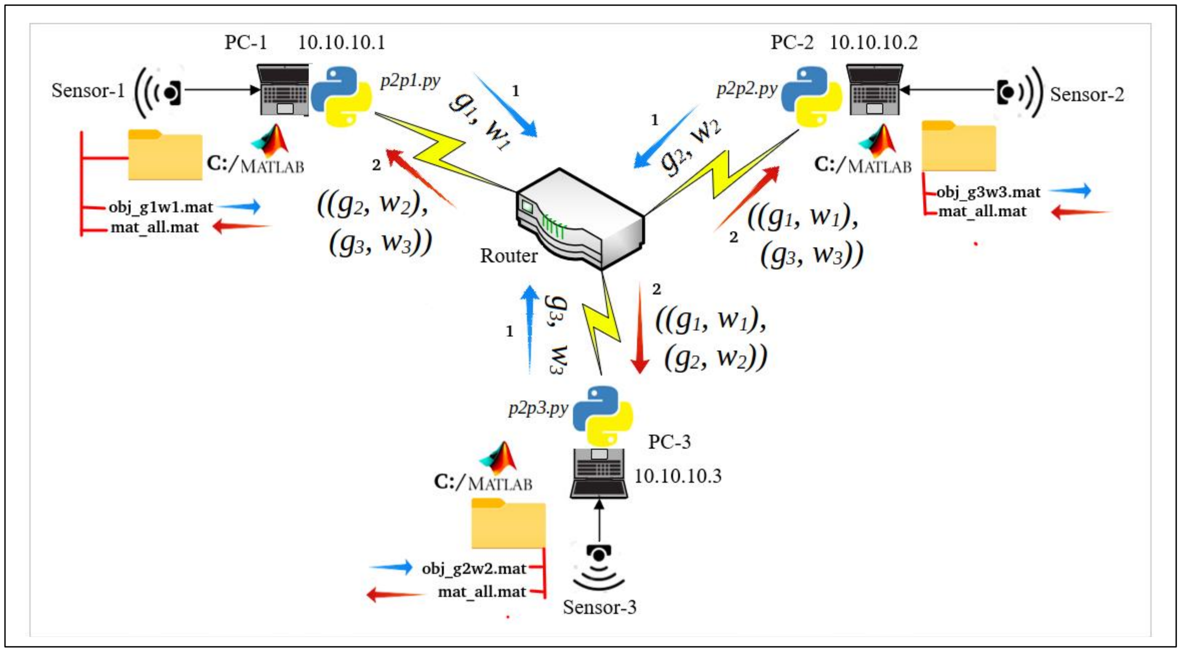
Network Communication Algorithm for Implementing DC-OMP-TA



Developed By	Alem Fitwi, PhD Student (Computer Engineering)
Adviser	Yu Chen, PhD
Department	Computer Engineering Stream, Department of Electrical & Computer Engineering
College	Watson School of Engineering,
University	Binghamton University, SUNY
Address	85 Murray Hill Rd, Vestal, NY 13850, USA
Duration	August 27 – October 15, 2021

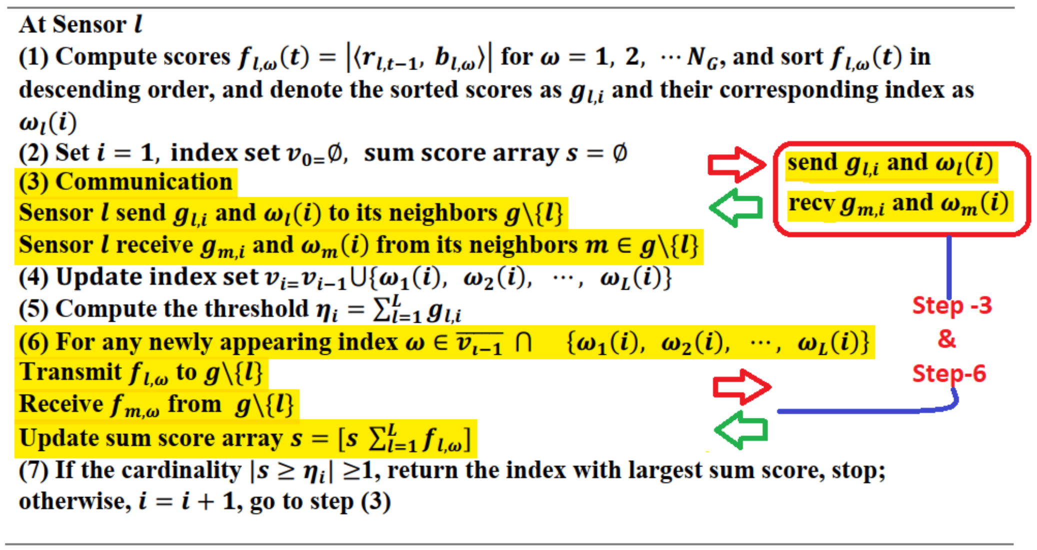
Binghamton, New York, USA

Contents

1. **Brief Project Description** 2
2. **Strategies Pursued to solve the Problem** 5
3. **Algorithm for Communication of Peers in DC-OMP-TA** 7
   - 3.1 **Terminologies and Definitions** 7
   - 3.2 **Proposed Algorithm (Algorithm-1)** 8
4. **Python Codes for Nodes/Peers Communication** 9
   - 4.1 **Requirements (requirements.txt)** 9
   - 4.2 **dcomptaconfigp2p1.py** 10
   - 4.3 **p2p1.py** 12
   - 4.4 **dcomptaconfigp2p2.py** 21
   - 4.5 **p2p2.py** 23
   - 4.6 **dcomptaconfigp2p3.py** 30
   - 4.7 **p2p3.py** 31
5. **MATLAB Functions for Interfacing** 39
   - 5.1 **send_and_recvp31.m** 39
   - 5.2 **send_and_recvp32.m** 40
   - 5.3 **send_and_recvp33.m** 41
   - 5.4 **send_and_recvp61.m** 43
   - 5.5 **send_and_recvp62.m** 44
   - 5.6 **send_and_recvp63.m** 46
6. **File Organization** 47
   - 6.1 **File Organization @PC-1** 48
   - 6.2 **File Organization @PC-2** 48
   - 6.3 **File Organization @PC-3** 49
7. **Test Results** 50
   - 7.1 **Test on a Single Machine using Multiple CMDs** 51
   - 7.2 **Test Based-on Three Separate Networked Virtual Machines** 54
8. **Conclusions** 55

1. Brief Project Description

As portrayed in Fig 1, the main goal of this project is developing and implement an algorithm that will enable three or more computers to communicate one another and exchange objects (lists, tuples, array, matrixes, dictionaries, MAT-files, …) for distributed processing. Particularly, it enables 3 or more computers running the DC-OMP-TA to exchange objects at steps 3 and 6.



```
At Sensor $l$
(1) Compute scores $f_{l,\omega}(t) = |\langle r_{l,t-1}, b_{l,\omega}\rangle|$ for $\omega = 1, 2, \cdots N_G$, and sort $f_{l,\omega}(t)$ in descending order, and denote the sorted scores as $g_{l,i}$ and their corresponding index as $\omega_l(i)$
(2) Set $i = 1$, index set $v_{0=}\emptyset$, sum score array $s = \emptyset$
(3) Communication
Sensor $l$ send $g_{l,i}$ and $\omega_l(i)$ to its neighbors $g\backslash\{l\}$
Sensor $l$ receive $g_{m,i}$ and $\omega_m(i)$ from its neighbors $m \in g\backslash\{l\}$
(4) Update index set $v_{i=}v_{i-1}\cup\{\omega_1(i), \omega_2(i), \cdots, \omega_L(i)\}$
(5) Compute the threshold $\eta_i = \sum_{l=1}^{L} g_{l,i}$
(6) For any newly appearing index $\omega \in \overline{v_{i-1}} \cap \{\omega_1(i), \omega_2(i), \cdots, \omega_L(i)\}$
Transmit $f_{l,\omega}$ to $g\backslash\{l\}$
Receive $f_{m,\omega}$ from $g\backslash\{l\}$
Update sum score array $s = [s \ \sum_{l=1}^{L} f_{l,\omega}]$
(7) If the cardinality $|s \geq \eta_i| \geq 1$, return the index with largest sum score, stop; otherwise, $i = i + 1$, go to step (3)
```

send $g_{l,i}$ and $\omega_l(i)$
recv $g_{m,i}$ and $\omega_m(i)$
Step -3
&
Step-6

Fig 1: DC-OMP-TA Algorithm

At step-3 of the DC-OMP-TA, the PCs exchange their ordered scores ($g_{l,i}$) and corresponding set of indices ($w_{l,i}$). For the sake of demonstration, dummy objects are provided in Fig 2.

@ PC-1

$$B_1 = \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix}, \ y_1 = \begin{pmatrix} 2 \\ 5 \\ 0 \end{pmatrix}, \ \& \ x_1 = \begin{pmatrix} x_{1,1} \\ x_{1,2} \\ x_{1,3} \end{pmatrix}$$

@ PC-2

$$B_2 = \begin{pmatrix} b_{2,1} & b_{2,2} & b_{2,3} \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}, \ y_2 = \begin{pmatrix} 2 \\ 3 \\ 5 \end{pmatrix}, \ \& \ x_2 = \begin{pmatrix} x_{2,1} \\ x_{2,2} \\ x_{2,3} \end{pmatrix}$$

@ PC-3

$$B_3 = \begin{pmatrix} b_{3,1} & b_{3,2} & b_{3,3} \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{pmatrix}, \ y_3 = \begin{pmatrix} 0 \\ 3 \\ 5 \end{pmatrix}, \ \& \ x_3 = \begin{pmatrix} x_{3,1} \\ x_{3,2} \\ x_{3,3} \end{pmatrix}$$

Fig 2 Sample Measurement Matrices and Signal Matrices @ PC-1, PC-2, & PC-3

DC-OMP-TA based Solution:

(1) Initialize $t = 1,\ \hat{u}_l(0) = \emptyset,\ x_l = 0,\ \&\ r_{l,0} = y_l$

@ PC-1: $t = 1,\ \hat{u}_1(0) = \emptyset,\ \&\ r_{1,0} = y_1$

@ PC-2: $t = 1,\ \hat{u}_2(0) = \emptyset,\ \&\ r_{2,0} = y_2$

@ PC-3: $t = 1,\ \hat{u}_3(0) = \emptyset,\ \&\ r_{3,0} = y_3$

(2) Compute & sort the scores & index sets of each node:

Table-1: Scores, Ordered Scores, and Index Sets At Each One of three computers

@ Computer	f-scores (dictionary)	g-scores (list)	W (Index Set)
PC-1	{0:2, 1:7, 2:5}	[7, 5, 2]	{1, 2, 0}
PC-2	{0:8, 1:7, 2:10}	[10, 8, 7]	{2, 0, 1}
PC-3	{0:8, 1:5, 2:0}	[8, 5, 0]	{0, 1, 2}

(3) Set $i = 1,\ index\ set\ v_0 = \emptyset,\ \&\ sum\ scores\ array\ s = \emptyset$

```
i, v0, s = 1, [], []
```

(4) Communication:

- **(4.1) Transmit**:
  - Node PC_1 sends g_1 to its neighbors PC_2 and PC_3;
  - Node PC_2 sends g_2 to its neighbors PC_1 and PC_3, and
  - Node PC_3 sends g_3 to its neighbors PC_1 and PC_2,
- **(4.2) Receive**:
  - Node PC_1 receives g_2 & g_3 from its neighbors PC_2 and PC_3, respectively;
  - Node PC_2 receives g_1 & g_3 from its neighbors PC_1 and PC_3, respectively; and
  - Node PC_3 receives g_1 & g_2 from its neighbors PC_1 and PC_2, respectively

Finally:

@PC-1:

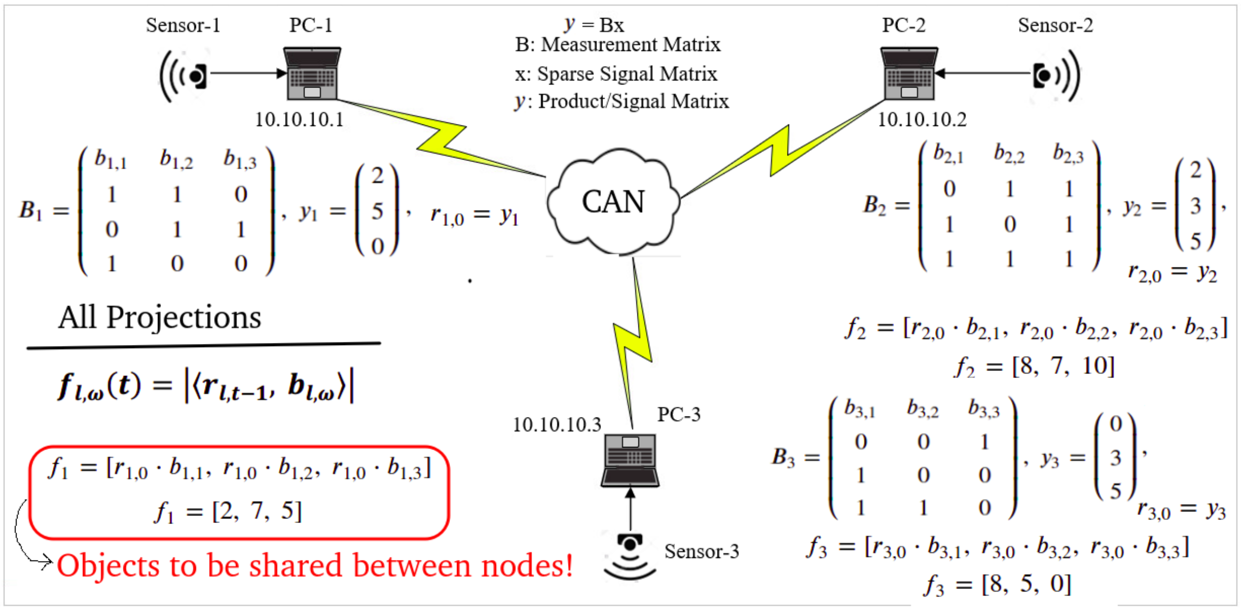
$$
\begin{aligned}
g = \{\ & g1 : \{1 : 7, 2 : 5, 0 : 2\}, \\
& g2 : \{2 : 10, 0 : 8, 1 : 7\}, \\
& g3 : \{0 : 8, 1 : 5, 2 : 0\} \\
\}
\end{aligned}
$$

@PC-2:

$$
\begin{aligned}
g = \{\ & g2 : \{2 : 10, 0 : 8, 1 : 7\}, \\
& g1 : \{1 : 7, 2 : 5, 0 : 2\}, \\
& g3 : \{0 : 8, 1 : 5, 2 : 0\} \\
\}
\end{aligned}
$$

@PC-3:

$$
\begin{aligned}
g = \{\ & g3 : \{0 : 8, 1 : 5, 2 : 0\}, \\
& g1 : \{1 : 7, 2 : 5, 0 : 2\}, \\
& g2 : \{2 : 10, 0 : 8, 1 : 7\} \\
\}
\end{aligned}
$$



$$B_1 = \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix},\ y_1 = \begin{pmatrix} 2 \\ 5 \\ 0 \end{pmatrix},\ r_{1,0} = y_1$$

$$f_{l,\omega}(t) = |\langle r_{l,t-1}, b_{l,\omega} \rangle|$$

$$f_1 = [r_{1,0} \cdot b_{1,1},\ r_{1,0} \cdot b_{1,2},\ r_{1,0} \cdot b_{1,3}]$$
$$f_1 = [2,\ 7,\ 5]$$

$$B_2 = \begin{pmatrix} b_{2,1} & b_{2,2} & b_{2,3} \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix},\ y_2 = \begin{pmatrix} 2 \\ 3 \\ 5 \end{pmatrix},\ r_{2,0} = y_2$$

$$f_2 = [r_{2,0} \cdot b_{2,1},\ r_{2,0} \cdot b_{2,2},\ r_{2,0} \cdot b_{2,3}]$$
$$f_2 = [8,\ 7,\ 10]$$

$$B_3 = \begin{pmatrix} b_{3,1} & b_{3,2} & b_{3,3} \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{pmatrix},\ y_3 = \begin{pmatrix} 0 \\ 3 \\ 5 \end{pmatrix},\ r_{3,0} = y_3$$

$$f_3 = [r_{3,0} \cdot b_{3,1},\ r_{3,0} \cdot b_{3,2},\ r_{3,0} \cdot b_{3,3}]$$
$$f_3 = [8,\ 5,\ 0]$$

Fig 3 Computation and Exchange of f-scores

Implementation of the "Communication Step" of the DC-OMP-TA is the main goal of this project, as depicted in Fig 3. As a result, we have developed a set of python applications and MATLAB functions (both work in unison) that enable computers to exchange the ordered scores (g) and corresponding index set (w) every time they reach at the communication steps of DC-OMP-TA.

2. Strategies Pursued to solve the Problem

We know that MATLAB is a powerful multi-paradigm programming language and numeric computing environment developed by MathWorks that allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, et cetera. However, MATLAB is proprietary programming language and everything is not an open-book like in other programming languages. It is true that MATLAB supports great parallel/cluster processing but you have to buy the licenses. Unlike other programming languages (like python), there is no accessible/open way where we can build sophisticated multiclient-server or peer-to-peer application for meaningful distributed computing using MATLAB. It supports a simple client-server application (licensed under Instrument Toolbox); but it is not possible to build sophisticated network application using MATLAB as far as we are concerned. We have explored and researched many materials and forums to verify whether it is possible to build a complicated network communication application using MATLAB or not; well, the answer is "it is not possible".

Then, we had to come up with another effective solution developed based on three strategies concisely described in Table 2 below. It requires interfacing between MATLAB & Python!

Table 2: Strategies based on which a solution for the DC-OMP-TA Scores Exchange was Designed.

Strategy	**Terse Description**
Strategy-1	Develop Python-based peer/client/Server Apps for Communication, refer to Fig 4
Strategy-2	Exchange only MATLAB & Python objects with added headers, refer to Fig 4
Strategy-3	Develop two MATLAB functions (one for step-3 & the other for step-6) for interfacing Python Apps & main MATLAB programs, refer to Fig 4

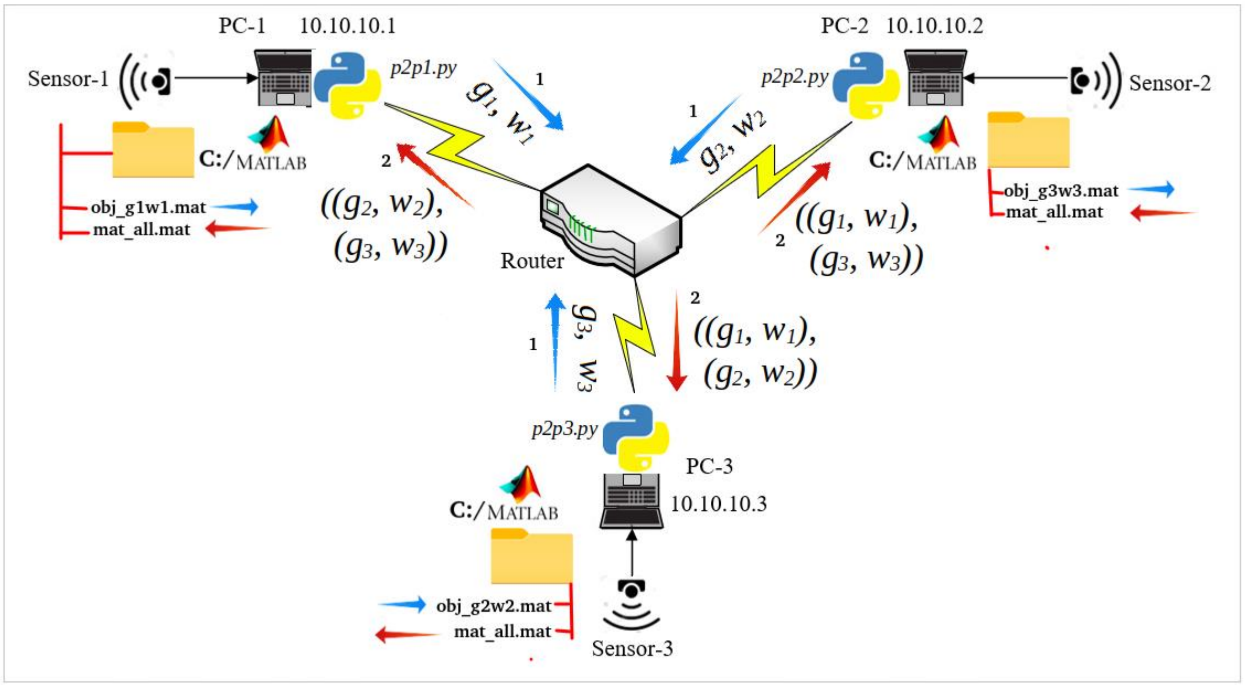


Figure 4: An Architecture of the Proposed Communication Algorithm for DC-OMP-TA

As clearly portrayed in Fig 4, three peer/client applications are developed by using Python 3.9 for the exchange of objects (matrices, arrays, lists, or dictionaries, …) between the three computers. One of them is designed to act as a server and peer to manage the communication establishment and object assembling.

@ ***Strategy-1***:

- The python apps enable the three PCs establish reliable communication using TCP/IP Sockets, as depicted in Fig 4.
- Once the communication is established, the server/peer runs forever and it does two main things: 1) it continually checks for a local MATLAB object (in the MATLAB Work Space on the machine it is running) if it is ready to be sent to the other PCs, and 2) it continuously listens for incoming data from other peers.
- Likewise, the peer/client apps also loop for eternity. They continuously check if the main MATLAB Code has reached at step-3 or step-6 and created objects that need to be transmitted to other remote peers.
- Tasks are carried out in a multi-threaded fashion!

@ ***Strategy-2***:

- MATLAB objects (matrix g, and index set w) are then exchanged as illustrated in Fig 4. The Python codes fetch such objects from the MATLAB Working Space and forward them to the other remote peers. They are both read and written in the form of a dictionary of dictionaries or in MAT-file format (*.mat).
- The MAT-objects are pickled or serialized into a form understandable by the TCP/IP Socket Protocols before transmission. This is, literally, a serialization process.
- Following a successful pickling process, a header is added to the objects for meaningful communication. A 20 Byte header information is added to every object transferred!

Device ID	**Step (3 or 6)**	**Actual Object Length After Pickling**
8 Bytes	5 Bytes	7 Bytes
• 'obj_g1w1', • 'obj_g2w2', or • 'obj_g3w3'	'step3', or 'step6'	On my PC, I am using a max TCP Recv Buffer size of 1048576 Bytes. *Never forget to increase the object length if your buffer is larger!*

- If your operating system is Linux, please use one of the following commands to check the minimum, default, and maximum TCP Receive Buffer Sizes on your machine. You can increase or decrease the size using system setting. Please note that you can't receive objects/data whose size is greater than the Recv Buffer. The Recv Buffer Size decides the maximum size of objects or data you can receive in a single communication!

TCP Receive Buffer Size (Min Default Max) in Bytes

```
$ cat /proc/sys/net/ipv4/tcp_rmem
$ sudo sysctl net ipv4.tcp_rmem
```

TCP Send Buffer Size (Min Default Max) in Bytes *$ cat /proc/sys/net/ipv4/tcp_wmem* *$ sudo sysctl net ipv4.tcp_wmem*		
Minimum	Default	Maximum
4096B	131072B	6291456B

- At step 3 (Fig 1), as indicated by the DC-OMP-TA algorithm, every node must send objects to its neighbors and receive objects from its neighbors before it can proceed to step-4. They have to wait on one another!
- As step 6 (Fig 1), if nodes have any new update, they send it to their neighbors; otherwise, they send only a "False" flag to indicate they have no update. No wait is required here!

@ Strategy-3:

- A MATLAB function that interfaces the main MATLAB Code and the python app at step-3 is developed. When the MATLAB control transfer reaches at step-3, this function does the following.
    - Cleans stale local and global objects saved in-the-just-previous step
    - Saves the objects w and g to work space: save ("obj_w1g1.mat", "g1", "w1")
    - Sends a read-trigger to the corresponding python app to inform it that local object is ready.
    - Waits in a while loop until pertinent results are returned
- A MATLAB function that interfaces the main MATLAB Code and the python app at step-6 is developed. When the MATLAB control transfer reaches at step-6, this function checks for new updates. If there exists any update, it sends an update-trigger to the python code. Then, the update is forwarded to its neighbors. Likewise, updates are received from its neighbors if they exist.
- Both of these MATLAB functions will be appropriately placed inside the main MATLAB program of the DC-OMP-TA, at the points where step-3 and step-6 ends, respectively.

3. Algorithm for Communication of Peers in DC-OMP-TA

3.1 Terminologies and Definitions

Terms	***Brief Description***
Socket	✓ Sockets are interior endpoints built for sending and receiving data ✓ A single network will have two sockets ✓ Sockets are a combination of an IP address and a port. ✓ A single device can have n-number of sockets based on the number of ports employed. ✓ Different ports are available for different protocols
socket.socket()	used to create sockets (required on both server and client to create sockets)
socket.accept():	used to accept a connection. It returns a pair of values (conn, address)

socket.bind():	Used to bind to the address that is specified as a parameter
socket.close():	Used to mark the socket as closed
socket.connect():	Used to connect to a remote address specified as the parameter.
socket.listen():	Used to enable the server to accept connections.
Objects	Lists, Tuples, Sets, Arrays, Matrices, Dictionaries, or MAT-files
Multi-threading	Supports handling of multiple (at least three) clients/peers at the same time, or simultaneously.

3.2 Proposed Algorithm (Algorithm-1)

DC-OMP-TA Communication Algorithm, @PC$_1$ (the same thing applies for other PCs, as well)

Steps **Description**

(1) Sends its *Device_ID* (*obj_g1w1*) and its IP Address to the server (if itself is not a server). Other peers do the same thing and communication is established. They all enter into listening mode! They listen for both local triggering signal and incoming objects from other peers.

Device IDs:

- @ PC$_1$: obj_g1w1
- @ PC$_2$: obj_g2w2
- @ PC$_3$: obj_g3w3

Python Programs:

- @ PC$_1$: p2p1.py
- @ PC$_2$: p2p2.py
- @ PC$_3$: p2p3.py

(2) When main control of the main MATLAB program reaches at *step-3*, the MATLAB function (*send_and_recvp31.m*) clears stale objects, saves new objects (g_1 & w_1) to work space, and triggers the python app running on PC_1 to read and transmit these objects to other peers.

MATLAB Functions @ each PC, for handling *step-3* of DC-OMP-TA

- @ PC$_1$: send_and_recvp31.m
- @ PC$_2$: send_and_recvp32.m
- @ PC$_3$: send_and_recvp33.m

Triggers @ each PC, initiated at *step-3* of DC-OMP-TA

- @ PC$_1$: “step3triggerw1g1.txt”
- @ PC$_2$: “step3triggerw2g2.txt”
- @ PC$_3$: “step3triggerw3g3.txt”

(3) Wait in loop until ordered scores along with their index sets (($g2$, $w2$), & ($g3$, $w3$)) are received from all other peers (PC_2 & PC_3, *respectively*)

(4) Once the *send_and_recvp31.m* function fully receives the scores from all other peers, it exits out of the idle loop and hands control to the main MATLAB program of DC-OMP-TA.

(5) At *step-6*, the MATLAB function pertinent to PC_1 (*send_and_recvp61.m*), clears stale objects, saves newly available updates to work space & sends read-trigger signal to the python app running on the same machine. On completion, it proceeds to the next step (control is returned to the main MATLAB program).

MATLAB Functions @ each PC, for handling *step-6* of DC-OMP-TA

- @ PC_1: send_and_recvp61.m
- @ PC_2: send_and_recvp62.m
- @ PC_3: send_and_recvp63.m

Triggers @ each PC, made at *step-6* of DC-OMP-TA

- @ PC_1: “step6triggerw1f1.txt”
- @ PC_2: “step6triggerw2f2.txt”
- @ PC_3: “step6triggerw3f3.txt

(6) Keep on listening for read-trigger signal from *send_and _recvp31.m*

(7) Go to step-2

4. Python Codes for Nodes/Peers Communication

4.1 Requirements (requirements.txt)

A PC must have python 3.9 installed on it with the most recent versions of following modules and packages. What is more, the PC must have the most recent MATLAB software with licenses installed on it!

```
import os

import sys

import glob

from sys import path

import numpy as np

import socket

from scipy.io import loadmat, savemat

from pickle import dumps, loads

from _thread import *

from datetime import datetime as dtt

from datetime import date,timedelta
```

In addition, there shouldn't be any restriction on the network that prevents the peers from communicating.

4.2 dcomptaconfigp2p1.py

- This is a configuration python file for PC_1

```
#-----------------------------------------------------------
# Configuration File of Peer/Server
#-----------------------------------------------------------
#-----------------------------------------------------------
#!/usr/bin/env python3
"""
Created on Sunday September 26 15:04:58 2021
@author: alem fitwi
"""
#-----------------------------------------------------------
#-----------------------------------------------------------
# Relative path to configuration file, CONFIG_PATH
#-----------------------------------------------------------
CONFIG_PATH = "config_mdl_p2p1"

#-----------------------------------------------------------
# Input_output path setting, IO_PATH
#-----------------------------------------------------------
IO_PATH="/home/alem/Desktop/ntk/DC_OMP_TA_Project/matlab1/"
# IO_PATH = "D:\\DC_OMP_TA_Project\\" # for Windows OS

#-----------------------------------------------------------
# User-defined Message Header-size, HEADER_SIZE
#-----------------------------------------------------------
HEADER_SIZE = 20 #Bytes

#-----------------------------------------------------------
# Server Port Number (p2p1.py), SERV_PORT (<= 2^16-1)
#-----------------------------------------------------------
SERV_PORT = 55577

#-----------------------------------------------------------
# IPv4 Address of p2p1, SERV_IP
#-----------------------------------------------------------
SERV_IP = '127.0.0.1' #Change it to real IP Address!

#-----------------------------------------------------------
# List of IDs of the three Nodes, namely p2p1, p2p2, & p2p3
#-----------------------------------------------------------
NODES_ID = ['obj_g1w1', 'obj_g2w2', 'obj_g3w3']
```

```
CUR_NODE_ID = 'obj_g1w1'

#-----------------------------------------------------------
# List of Server-side DC-OMP-TA Keys
#-----------------------------------------------------------
SERV_KEYS3 = ['g1', 'w1']
SERV_KEYS6 = ['ff1', 'ww1']
#-----------------------------------------------------------
# Dictionary all possible keys of other peers
#-----------------------------------------------------------
KEYS_DICT_STEP3 = {'obj_g2w2': ['g2', 'w2'],
                   'obj_g3w3': ['g3', 'w3']}

KEYS_DICT_STEP6 = {'obj_g2w2': ['ff2', 'ww2'],
                   'obj_g3w3': ['ff3', 'ww3']}

#-----------------------------------------------------------
# Number of Bytes received in a single communication
#-----------------------------------------------------------
# Linux OS:
    # TCP Receive Buffer Size (Min Default Max) in Bytes
        # $ cat /proc/sys/net/ipv4/tcp_rmem
        # $ sudo sysctl net ipv4.tcp_rmem
    # TCP Send Buffer Size (Min Default Max) in Bytes
        # $ cat /proc/sys/net/ipv4/tcp_wmem
        # $ sudo sysctl net ipv4.tcp_wmem
RECV_BYTES = 1048576 # Bytes

#-----------------------------------------------------------
# Template For Aggregated .mat file sent back to all peers
#-----------------------------------------------------------
DICTGW = {"Project":"DC_OMP_TA","By":"Alem Fitwi & Yu Chen",
        "error":"No Error!",
         "g1": '', 'w1': '', 'g2': '', 'w2': '', 'g3': '',
         "w3":''}
DICTFW = {"Project":"DC_OMP_TA","By":"Alem Fitwi & Yu Chen",
        "error":"No Error!"}

#-----------------------------------------------------------
# Step-3 & Step-7 Read-Trigger Filenames
#-----------------------------------------------------------
TRIGGERS = ['step3triggerw1g1.txt','step6triggerw1f1.txt']

#-----------------------------------------------------------
# Object names for Step-3 & Step-6 of DC-OMP_TA
```

```
#----------------------------------------------------------
MAT_FILE_NAME3 = ["obj_g1w1", "obj_mat_all", "DICTGW.mat",
                   "error_msg.mat"]
MAT_FILE_NAME6 = ["obj_f1w1", "obj_mat_fs", "DICTFW.mat"]

#----------------------------------------------------------
# Names of Pertinent DC-OMP_TA Steps
#----------------------------------------------------------
TA_STEPS = ['step3', "step6"]

#----------------------------------------------------------
# --------------------------END---------------------------
#----------------------------------------------------------
```

4.3 p2p1.py

- This is a server/peer python program that runs on PC_1

```
#----------------------------------------------------------
# Project_name: DC-OMP-TA Communications @ Step-3 & Step-6
# -----------------Server/Peer----------------------------
#----------------------------------------------------------
#!/usr/bin/env python3
"""
Created on Saturday September 25 14:12:27 2021
@author: alem fitwi
"""


#----------------------------------------------------------
# Import Necessary Libraries and Packages
#----------------------------------------------------------
import os
import sys
import glob
from sys import path
import numpy as np
import socket
from scipy.io import loadmat, savemat
from pickle import dumps, loads
from _thread import *
from datetime import datetime as dtt
from datetime import date,timedelta
#----------------------------------------------------------
# Global Variables
#----------------------------------------------------------
```

```python
ThreadCount = 0
step3ID_lst, step6ID_lst, connections = [], [], []

#-----------------------------------------------------------
# Import All Necessary Constants, config files and Paths
#-----------------------------------------------------------
COM_PATH = "/home/alem/Desktop/ntk/DC_OMP_TA_Project/"
path.append(os.path.join(COM_PATH,'config_mdl_p2p1/'))

from dcomptaconfigp2p1 import HEADER_SIZE, IO_PATH,\
    SERV_PORT,SERV_IP, NODES_ID, RECV_BYTES, DICTGW,\
    DICTFW, CUR_NODE_ID, TRIGGERS, SERV_KEYS3,\
    KEYS_DICT_STEP3, KEYS_DICT_STEP6, MAT_FILE_NAME3,\
    MAT_FILE_NAME6, TA_STEPS, SERV_KEYS6
#-----------------------------------------------------------
# Extract keys of (ID, [g,w]) pairs of server
keys3 =  list(KEYS_DICT_STEP3.keys())
keys6 =  list(KEYS_DICT_STEP6.keys())

#-----------------------------------------------------------
# Read .mat file from IO_PATH, to be sent to remote peers
#-----------------------------------------------------------
def read_mat(IO_PATH, mat_name):
    filename = mat_name  + ".mat"
    try:
        path1 = os.path.join(
              IO_PATH, filename)
        mat = loadmat(path1)
    except:
        return False
    else:
        return mat

#-----------------------------------------------------------
# Write .mat files fetched from remote peers to IO_PATH
#-----------------------------------------------------------
def save_mat(IO_PATH, mat_name, matobject):
    filename = mat_name + ".mat"
    try:
        path1 = os.path.join(
                    IO_PATH, filename)
        savemat(path1, matobject)
    except:
        return False
    else:
        return True

#-----------------------------------------------------------
# Serialize Objects using pickle.dumps
#-----------------------------------------------------------
def serialize(obj):
    try:
        ser = dumps(obj)
    except:
        return False
    else:
        return ser

#-----------------------------------------------------------
# Deserialize Objects using pickle.loads
#-----------------------------------------------------------
def deserialize(obj):
    try:
        deser = loads(obj)
    except:
        return False
    else:
        return deser

#-----------------------------------------------------------
# Receive data from another peer
#-----------------------------------------------------------
def recv_obj(conn, RECV_BYTES, HEADER_SIZE):
    """Receives objects from
       other peers"""

    full_msg = b''
    while True:
        pyobj = conn.recv(RECV_BYTES)
        print("Recv: ",pyobj[:HEADER_SIZE])
        msg_len = int(pyobj[:HEADER_SIZE])
        full_msg += pyobj
        if len(full_msg) - HEADER_SIZE == msg_len:
            infos = full_msg[HEADER_SIZE:HEADER_SIZE+13
                     ].decode("utf-8")
            idd = infos[:8]
            step = infos[8:].strip()
            break
    return idd, step, deserialize(full_msg[HEADER_SIZE+13:])
```

```
#----------------------------------------------------------
# Send data to all other peers --> ['obj_g2w2', 'obj_g3w3']
#----------------------------------------------------------
def send_obj(conn, obj):
    obj1 = serialize(obj)
    obj3 =  bytes(f"{len(obj1):<{HEADER_SIZE}}",
                      "utf-8")+obj1
    conn.sendall(obj3)

#----------------------------------------------------------
# Check steps-3&6 .mat read-trigger in IO_PATH Continually!
#----------------------------------------------------------
def isTrigger(path2, TRIGGERS):
    fstep3 = os.path.join(path2,TRIGGERS[0])
    fstep6 = os.path.join(path2,TRIGGERS[1])
    bstep3 = os.path.exists(fstep3)
    bstep6 = os.path.exists(fstep6)
    return bstep3, bstep6

#----------------------------------------------------------
# A Function that removes a file from disk
#----------------------------------------------------------
def remove_file(path2, filename):
    try:
        path1 = os.path.join(path2, filename)
        file = glob.glob(path1)
        os.remove(file[0])
    except:
        print(sys.exc_info()[0])

#----------------------------------------------------------
# A Function that grabs current time and date
#----------------------------------------------------------
def get_datetime():
    tday = dtt.now()
    day = tday.strftime("%A")[:3]
    mon = tday.strftime("%B")[:3]
    time = tday.strftime("%d %H:%M:%S %Y")
    return day+' '+mon+'  '+time

#----------------------------------------------------------
# A Function that updates the date of obj.mat object
#----------------------------------------------------------
def update_date(obj):
    datetime = get_datetime()
    tmp  = obj['__header__']
    tmp= tmp.decode("utf-8").split('Created on:')
    tmp = tmp[0]+'Created on: '+datetime
    obj['__header__'] = bytes(tmp,"utf-8")
    return obj

#------------------------------------------------------------
# Multithreading peers, ['obj_g1w1', 'obj_g2w2', 'obj_g3w3']
#------------------------------------------------------------
def parallelize_peers(conn, obj_mat_all, obj_mat_fs, error,
    COM_PATH, IO_PATH, TRIGGERS, CUR_NODE_ID, HEADER_SIZE,
    keys3,keys6,TA_STEPS,MAT_FILE_NAME3, MAT_FILE_NAME6,
    KEYS_DICT_STEP3, KEYS_DICT_STEP6):
    #--------------------------------------------------------
    global connections
    global step3ID_lst
    global step6ID_lst
    #--------------------------------------------------------
    while True:
        #----------------------------------------------------
        if conn not in connections:
            connections.append(conn)
            idd, step, pyobj = recv_obj(conn, RECV_BYTES,
                                        HEADER_SIZE)

        #----------------------------------------------------
            if step == TA_STEPS[0]:
                which_step = TA_STEPS[0]

                try:
                    tmp3 = KEYS_DICT_STEP3[idd]
                    for val3 in tmp3:
                        obj_mat_all[val3] = pyobj[val3]

                    if idd not in step3ID_lst:
                        step3ID_lst.append(idd)
                    print("End of for if")
                except:
                    err_msg = sys.exc_info()[0]
                    err_msg1 = "At server: "+str(err_msg)
                    print(err_msg1)
                    obj_mat_all['error'] = "At server: "+str(err_msg)

                    tmpv = "Error: please check the .mat files of all "
                    obj_mat_all['error2'] = tmpv + "nodes are good!"
                    break

                logics3 = []
                for k3 in keys3:
                    if k3 in step3ID_lst:
                        logics3.append(True)
                if all(logics3) and len(logics3)==len(keys3):
                    break
        #--------------------------------------------------
            if step == TA_STEPS[1]:
                which_step = TA_STEPS[1]

                try:
                    if bool(pyobj['flag']) == True:
                        tmp6= KEYS_DICT_STEP6[idd]
                        for val6 in tmp6:
                            obj_mat_fs[val6] = pyobj[val6]


                    if idd not in step6ID_lst:
                        step6ID_lst.append(idd)
                except:
                    err_msg = sys.exc_info()[0]
                    err_msg1 = "At server: "+str(err_msg)
                    print(err_msg1)
                    obj_mat_fs['error'] = err_msg1
                    tmpv = "Error: please check the .mat files of all "
                    obj_mat_fs['error2'] = tmpv + "nodes are good!"
                    break

                logics6 = []
                for k6 in keys6:
                    if k6 in step6ID_lst:
                        logics6.append(True)
                if all(logics6) and len(logics6)==len(keys6):
                    break
        #--------------------------------------------------
    #------------------------------------------------------
    which_step = ''
    while True:
        #--------------------------------------------------
        bstep3, bstep6 = isTrigger(IO_PATH, TRIGGERS)
        if bstep3 == True:
            which_step = TA_STEPS[0]
            obj_g1w1 = read_mat(IO_PATH, CUR_NODE_ID)
            if isinstance(obj_g1w1, bool) == True:
                continue
            else:
                for skey3 in SERV_KEYS3:
                    obj_mat_all[skey3] = obj_g1w1[skey3]

                save_mat(IO_PATH, MAT_FILE_NAME3[1],
                                               obj_mat_all)
                filename = CUR_NODE_ID +'.mat'
                remove_file(IO_PATH, filename)
            remove_file(IO_PATH, TRIGGERS[0])
            break
        #-------------------------------------------------------
        elif bstep6 == True and bstep3 == False:
            which_step = TA_STEPS[1]
            obj_f1w1 = read_mat(IO_PATH, MAT_FILE_NAME6[0])
            if isinstance(obj_f1w1, bool) == True:
                continue
            else:
                if bool(obj_f1w1['flag']) == True:
                    for skey6 in SERV_KEYS6:
                        obj_mat_fs[skey6]=obj_f1w1[skey6]

                save_mat(IO_PATH, MAT_FILE_NAME6[1],
                                              obj_mat_fs)
                filename = "obj_f1w1" +'.mat'
                remove_file(IO_PATH, filename)
            remove_file(IO_PATH, TRIGGERS[1])
            break
        else:
            error2="No Triggers or both are simultaneously"
            obj_mat_fs['error2'] = error2 + " set!"

    #-------------------------------------------------------
    # Send objects to peers
    #-------------------------------------------------------
    if which_step == TA_STEPS[0]:
        for conn3 in connections:
            send_obj(conn3, obj_mat_all)

    #-------------------------------------------------------
    if which_step == TA_STEPS[1]:
        for conn6 in connections:
            send_obj(conn6, obj_mat_fs)
    #-------------------------------------------------------
```

```
    # Reset Global Variables for next connections
    step3ID_lst, step6ID_lst, connections = [], [], []

    #-------------------------------------------------------
    # Close current sessions
    conn.close()

#-----------------------------------------------------------
# The Main Function where every other function is run!
#-----------------------------------------------------------
if __name__ == '__main__':
    #Create A Server/Peer Socket
    ServerSocket = socket.socket(socket.AF_INET,
                                 socket.SOCK_STREAM)
    #-------------------------------------------------------
    # Bind Server IP Address and Port Number
    try:
        ServerSocket.bind((SERV_IP, SERV_PORT))
    except socket.error as e:
        print(str(e))
    #-------------------------------------------------------
    print('**********************************************')
    print('Waiting For Connection Requests From Peers ...')
    print('**********************************************')
    ServerSocket.listen(5) # Listening for other peers
    #-------------------------------------------------------
    initpath = os.path.join(COM_PATH, 'config_mdl_p2p1/')
    error = loadmat(initpath + MAT_FILE_NAME3[3])
    #-------------------------------------------------------
    savemat(initpath+MAT_FILE_NAME3[2], DICTGW)
    obj_mat_all = loadmat(initpath+MAT_FILE_NAME3[2])
    print("---------------------------------------")
    print(f"At Start, obj_mat_all: {obj_mat_all}")
    print("---------------------------------------")

    savemat(initpath+MAT_FILE_NAME6[2], DICTFW)
    obj_mat_fs = loadmat(initpath+MAT_FILE_NAME6[2])
    print("---------------------------------------")
    print(f"At Start, obj_mat_fs: {obj_mat_fs}")
    print('**********************************************')
    #-------------------------------------------------------
    rounds = 1
    while True:
    #-------------------------------------------------------
```

```
        print('******************************************')
        print(f"@ Round: {rounds}")
        print('******************************************')
        # Read Base Object File for Step-3 of TA
        obj_mat_all = update_date(obj_mat_all)
        print('******************************************')
        print(f"In A While Loop obj_mat_all: {obj_mat_all}")
        print('******************************************')
    #------------------------------------------------------
        # Read Base Object File for Step-6 of TA
        obj_mat_fs = update_date(obj_mat_fs)
        print('******************************************')
        print(f"In A While Loop obj_mat_fs: {obj_mat_fs}")
        print('******************************************')

        # Accept Multiple Connection Requests
        print('******************************************')
        print('Listening ...')
        print('------------------------------------------')
        conn, address = ServerSocket.accept()
        print(f"Connected To Peer {address} ...")
        print('******************************************')

        print('******************************************')
        print(f"Organizing .mat Objects In Parallel ...")
        start_new_thread(parallelize_peers,
           (conn, obj_mat_all, obj_mat_fs, error,
           COM_PATH, IO_PATH, TRIGGERS, CUR_NODE_ID,
           HEADER_SIZE, keys3,keys6,TA_STEPS,MAT_FILE_NAME3,
           MAT_FILE_NAME6, KEYS_DICT_STEP3,
           KEYS_DICT_STEP6))
        print('******************************************')

        print('******************************************')
        # Count Threads and Display the count
        print(f" Number Of Peers/Threads Running ...")
        ThreadCount += 1
        print('Thread Number: ' + str(ThreadCount))
        print('******************************************')

        # Reset Count of Threads if > 2^{16}-1
        if ThreadCount > 65535:
            ThreadCount = 0

        # Reset rounds count if > 2^{16}-1
        if rounds > 65535:
            rounds = 0
        rounds += True

    #-------------------------------------------------------
    #Close Server Socket
    ServerSocket.close()
    #-------------------------------------------------------
#-----------------------------------------------------------
# End of p2p1.py program, which acts as a server/peer!
#-----------------------------------------------------------
```

4.4 dcomptaconfigp2p2.py

£ This is a configuration python file for PC_2

```
#-----------------------------------------------------------
#!/usr/bin/env python3
"""
Created on Sunday September 26 18:37:33 2021
@author: alem fitwi
"""
#-----------------------------------------------------------
#-----------------------------------------------------------
# Relative path to configuration file, CONFIG_PATH
#-----------------------------------------------------------
CONFIG_PATH = "config_mdl_p2p2"

#-----------------------------------------------------------
# Input_output path setting,IO_PATH
#-----------------------------------------------------------
IO_PATH = "/home/alem/Desktop/ntk/DC_OMP_TA_Project/matlab2/"
# IO_PATH = "D:\\DC_OMP_TA_Project\\" # for Windows OS
#-----------------------------------------------------------
# User-defined Message Header-size, HEADER_SIZE
#-----------------------------------------------------------
HEADER_SIZE = 20 #Bytes

#-----------------------------------------------------------
# Server Port Number (p2p1.py), SERV_PORT (<= 2^16-1)
#-----------------------------------------------------------
SERV_PORT = 55577

#-----------------------------------------------------------
# IPv4 Address of p2p1, SERV_IP
#-----------------------------------------------------------
```

```
SERV_IP = '127.0.0.1' #Change it to real IP Address!

#-----------------------------------------------------------
# List of IDs of the three Nodes, namely p2p1, p2p2, & p2p3
#-----------------------------------------------------------
NODES_ID = ['obj_g1w1', 'obj_g2w2', 'obj_g3w3']
CUR_NODE_ID = 'obj_g2w2'
#-----------------------------------------------------------
# Number of Bytes received in a single communication
#-----------------------------------------------------------
# Linux OS:
    # TCP Receive Buffer Size (Min Default Max) in Bytes
        # $ cat /proc/sys/net/ipv4/tcp_rmem
        # $ sudo sysctl net ipv4.tcp_rmem
    # TCP Send Buffer Size (Min Default Max) in Bytes
        # $ cat /proc/sys/net/ipv4/tcp_wmem
        # $ sudo sysctl net ipv4.tcp_wmem
RECV_BYTES = 1048576 #Bytes

#-----------------------------------------------------------
# Step-3 & Step-7 Read-Trigger Filenames
#-----------------------------------------------------------
TRIGGERS = ['step3triggerw2g2.txt','step6triggerw2f2.txt']
             #step3triggerw2g2.txt
#-----------------------------------------------------------
# Object names for Step-3 & Step-6 of DC-OMP_TA
#-----------------------------------------------------------
MAT_FILE_NAME3 = ["obj_g2w2", "obj_mat_all"]
MAT_FILE_NAME6 = ["obj_f2w2", "obj_mat_fs"]

#-----------------------------------------------------------
# Names of Pertinent DC-OMP_TA Steps
#-----------------------------------------------------------
TA_STEPS = ['step3', "step6"]

#-----------------------------------------------------------
# --------------------------END---------------------------
#-----------------------------------------------------------
```

4.5 p2p2.py

- This is a client/peer python program that runs on PC_2

```
#------------------------------------------------------------
# Project_name: DC-OMP-TA Communications @ Step-3 & Step-6
# -----------------Peer/Client---------------------------
#------------------------------------------------------------
#!/usr/bin/env python3
"""
Created on Saturday September 25 18:37:33 2021
@author: alem fitwi
"""
#------------------------------------------------------------
# Import Necessary Libraries and Packages
#------------------------------------------------------------
import os
import sys
import glob
from sys import path
import numpy as np
import socket
from scipy.io import loadmat, savemat
from pickle import dumps, loads
from _thread import *

#------------------------------------------------------------
# Import All Necessary Constants, config files and Paths
#------------------------------------------------------------
# Set the main absolute path (excluding file name) here!
COM_PATH = "/home/alem/Desktop/ntk/DC_OMP_TA_Project/"
path.append(os.path.join(COM_PATH,'config_mdl_p2p2/'))

# Import all user-defined parameters and configurations
from dcomptaconfigp2p2 import HEADER_SIZE,IO_PATH,\
    SERV_PORT,SERV_IP, NODES_ID, RECV_BYTES,\
    CUR_NODE_ID, TRIGGERS, MAT_FILE_NAME3, \
    MAT_FILE_NAME6, TA_STEPS

#------------------------------------------------------------
# Read .mat file from IO_PATH, to be sent to remote peers
#------------------------------------------------------------
def read_mat(IO_PATH, mat_name):
    """Reads .mat MATLAB Objects to disk"""
    filename = mat_name + ".mat"
    try:
        path1 = os.path.join(
              IO_PATH, filename)
        mat = loadmat(path1)
    except:
        return False
    else:
        return mat

#-----------------------------------------------------------
# Write .mat files fetched from remote peers to IO_PATH
#-----------------------------------------------------------
def save_mat(IO_PATH, mat_name, matobject):
    """Saves .mat MATLAB Objects to disk"""
    filename = mat_name + ".mat"
    try:
        path1 = os.path.join(
                    IO_PATH, filename)
        savemat(path1, matobject)
    except:
        return False
    else:
        return True

#-----------------------------------------------------------
# Serialize Objects using pickle.dumps
#-----------------------------------------------------------
def serialize(obj):
    """Pickles objects into byte-series"""
    try:
        ser = dumps(obj)
    except:
        return False
    else:
        return ser

#-----------------------------------------------------------
# Deserialize Objects using pickle.loads
#-----------------------------------------------------------
def deserialize(obj):
    """unpickles objects into objects"""
    try:
        deser = loads(obj)
    except:
        return False
    else:
        return deser

#------------------------------------------------------------
# Receive data from another peer
#------------------------------------------------------------
def recv_obj(conn, RECV_BYTES, HEADER_SIZE):
    """Receives objects from other peers"""
    full_msg = b''
    while True:
        pyobj = conn.recv(RECV_BYTES)
        msg_len = int(pyobj[:HEADER_SIZE])
        full_msg += pyobj
        if len(full_msg) - HEADER_SIZE == msg_len:
            break
    return deserialize(full_msg[HEADER_SIZE:])

#------------------------------------------------------------
# Send data to all other peers --> ['obj_g2w2', 'obj_g3w3']
#------------------------------------------------------------
def send_obj(conn, obj, CUR_NODE_ID, step):
    """Sends pickled objects to other peers"""
    idd = CUR_NODE_ID + step
    obj1 = serialize(obj)
    obj1 = bytes(f"{idd}", "utf-8")+obj1
    obj3 =  bytes(f"{len(obj1):<{HEADER_SIZE}}",
                  "utf-8")+obj1
    conn.sendall(obj3)

#------------------------------------------------------------
# Check steps-3&6 .mat read-trigger in IO_PATH Continually!
#------------------------------------------------------------
def isTrigger(path2, TRIGGERS):
    """Checks step-3 & 6 Triggers"""
    fstep3 = os.path.join(path2,TRIGGERS[0])
    fstep6 = os.path.join(path2,TRIGGERS[1])
    bstep3 = os.path.exists(fstep3)
    bstep6 = os.path.exists(fstep6)
    return bstep3, bstep6

#------------------------------------------------------------
# A Function that removes a file from disk
#------------------------------------------------------------
def remove_file(path2, filename):
    """Removes files/objects from Disk"""
    try:
```

```python
        path1 = os.path.join(path2, filename)
        file = glob.glob(path1)
        os.remove(file[0])
    except:
        print(sys.exc_info()[0])

#----------------------------------------------------------
# The Main Function where every other function is run!
#----------------------------------------------------------
if __name__ == '__main__':
    """Calls and executes all other functions"""

    rn = 1 # Rounds counter
    while True:
        # Reset Counter
        if rn > 65535:
            rn =1
    #------------------------------------------------------
        print('******************************************')
        print('Waiting For A Trigger From MATLAB ...')
        print('---------------------------------------')
        bstep3, bstep6 = isTrigger(IO_PATH, TRIGGERS)
        print(f"At Round {rn}")
        print('******************************************')
    #------------------------------------------------------
        if bstep3 == True:
            print('**************************************')
            disp_msg0 = f"Processing Step-3 Of The "
            print(disp_msg0+f"DC_OMP_TA, @ Round {rn} ...")
            print('--------------------------------------')
            ClientSocket = socket.socket(socket.AF_INET,
                                         socket.SOCK_STREAM)
            print('**************************************')

    #------------------------------------------------------
            obj_g2w2 = read_mat(IO_PATH,MAT_FILE_NAME3[0])
            print("obj_g2w2", obj_g2w2)
            if isinstance(obj_g2w2, bool) == True:
                print('**********************************')
                print("No obj_g2w2.mat Object To Be Sent!")
                print('**********************************')
                continue
            else:
                print('**********************************')
                print('Local Object obj_g2w2 Read...')
                print('----------------------------------')
                print(f"obj_g2w2 = {obj_g2w2}")
                print('**********************************')
    #------------------------------------------------------
            try:
                ClientSocket.connect((SERV_IP, SERV_PORT))
            except socket.error as e:
                print(str(e))
    #------------------------------------------------------
            print('**************************************')
            disp_msg1 = "Waiting To Send Local Mat Object"
            print(disp_msg1 + " To Other Peers ...")
            print('--------------------------------------')
            send_obj(ClientSocket, obj_g2w2, CUR_NODE_ID,
                            TA_STEPS[0])
            print('--------------------------------------')
            print('obj_g2w2.mat Object Was Sent To Peers')
            print(obj_g2w2)
            print('**************************************')
    #------------------------------------------------------
            print('**************************************')
            disp_msg3 = "Waiting To Receive Objects "
            print(disp_msg3 + 'From Other Peers...')
            print('--------------------------------------')
            obj_mat_all = recv_obj(ClientSocket,RECV_BYTES,
                                              HEADER_SIZE)
            disp_msg3 = "obj_mat_all.mat Received From "
            print(disp_msg3 + ' Other Peers...')
            print('--------------------------------------')
            print(obj_mat_all)
            print('**************************************')

    #------------------------------------------------------
            print('**************************************')
            disp_msg4 = "Saving obj_mat_all.mat To "
            print(disp_msg4 + 'MATLAB Work Space...')
            print('--------------------------------------')
            save_mat(IO_PATH, MAT_FILE_NAME3[1],
                                               obj_mat_all)
            print("obj_mat_all Was Successfully Saved!")
            print('**************************************')
    #------------------------------------------------------
            print('**************************************')
            disp_msg5 = "Removing Triggers From MATLAB"
            print(disp_msg5+' Work Space...')
            print('--------------------------------------')
            remove_file(IO_PATH, TRIGGERS[0])
            remove_file(IO_PATH, MAT_FILE_NAME3[0]+".mat")
            trig = "Triggers/Inputs Were Successfully "
            print(trig + "Removed From Work Space")
            print('**************************************')
    #------------------------------------------------------
            print('**************************************')
            dp = f" @ Round {rn}"
            print(f"End Of Step-3 Of The DC_OMP_TA"+dp)
            print('--------------------------------------')

    #------------------------------------------------------
    #------------------------------------------------------
        elif bstep6 == True and bstep3 == False:
            print('**************************************')
            disp0 = f"Processing Step-6 Of The "
            print(disp0+f"DC_OMP_TA, @ Round {rn} ...")
            print('--------------------------------------')
            print('**************************************')
            print("Processing Step-3 Of The DC_OMP_TA ...")
            ClientSocket = socket.socket(socket.AF_INET,
                              socket.SOCK_STREAM)
            print('**************************************')
    #------------------------------------------------------
            obj_f2w2 = read_mat(IO_PATH,MAT_FILE_NAME6[0])
            if isinstance(obj_f2w2, bool) == True:
                print('**********************************')
                print("No obj_f2w2.mat Object To Be Sent!")
                print('**********************************')
                continue
            else:
                print('**********************************')
                print('Local Object obj_f2w2 Read...')
                print('----------------------------------')
                print(f"obj_f2w2 = {obj_f2w2}")
                print('**********************************')

    #------------------------------------------------------
            try:
                ClientSocket.connect((SERV_IP, SERV_PORT))
            except socket.error as e:
                print(str(e))
    #------------------------------------------------------
            print('**************************************')
```

```
            print('Waiting To Send Mat Object To Peers...')
            print('--------------------------------------')
            send_obj(ClientSocket, obj_f2w2, CUR_NODE_ID,
                            TA_STEPS[1])
            print('--------------------------------------')
            print('obj_f2w2.mat Object Sent To Peers...')
            print(obj_f2w2)
            print('--------------------------------------')
            print('**************************************')
    #------------------------------------------------------
            print('**************************************')
            disp_msg6 = "Waiting To Receive Objects "
            print(disp_msg6 + 'From Other Peers...')
            print('--------------------------------------')
            obj_mat_fs = recv_obj(ClientSocket, RECV_BYTES,
                                              HEADER_SIZE)
            print('--------------------------------------')
            print('obj_mat_fs Received From Other Peers!')
            print(obj_mat_fs)
            print('**************************************')

    #------------------------------------------------------
            print('**************************************')
            disp_msg7 = "Saving obj_mat_fs.mat To MATLAB"
            print(disp_msg7 + ' Work Space...')
            print('**************************************')
            save_mat(IO_PATH,MAT_FILE_NAME6[1],obj_mat_fs)
    #------------------------------------------------------
            print('**************************************')
            disp_msg8 = "Removing Triggers From MATLAB "
            print(disp_msg8 + 'Work Space ...')
            print('--------------------------------------')
            remove_file(IO_PATH, TRIGGERS[1])
            remove_file(IO_PATH, MAT_FILE_NAME6[0]+".mat")
            eras = "Triggers/Inputs Were Successfully"
            print(eras + " Removed From Work Space!")
            print('**************************************')

    #------------------------------------------------------
            print('**************************************')
            dp2 = f" @ Round {rn}"
            print(f"End Of step-6 Of The DC_OMP_TA"+dp2)
            print('**************************************')
    #------------------------------------------------------
        else:
            pass
        rn += True
    #------------------------------------------------------
    ClientSocket.close()

#----------------------------------------------------------
# End of p2p2.py program, which acts like a peer/client!
#----------------------------------------------------------
```

4.6 dcomptaconfigp2p3.py

£ This is a configuration python file for PC_3

```
#----------------------------------------------------------
#!/usr/bin/env python3
"""
Created on Sunday September 26 21:17:25 2021
@author: alem fitwi
"""
#----------------------------------------------------------
#----------------------------------------------------------
# Relative path to configuration file, CONFIG_PATH
#----------------------------------------------------------
CONFIG_PATH = "config_mdl_p2p3"

#----------------------------------------------------------
# Input_output path setting,IO_PATH
#----------------------------------------------------------
IO_PATH = "/home/alem/Desktop/ntk/DC_OMP_TA_Project/matlab3/"
# IO_PATH = "D:\\DC_OMP_TA_Project\\" # for Windows OS

#----------------------------------------------------------
# User-defined Message Header-size, HEADER_SIZE
#----------------------------------------------------------
HEADER_SIZE = 20 #Bytes

#----------------------------------------------------------
# Server Port Number (p2p1.py), SERV_PORT (<= 2^16-1)
#----------------------------------------------------------
SERV_PORT = 55577

#----------------------------------------------------------
# IPv4 Address of p2p1, SERV_IP
#----------------------------------------------------------
SERV_IP = '127.0.0.1' #Change it to real IP  Address
```

```
#-----------------------------------------------------------
# List of IDs of the three Nodes, namely p2p1, p2p2, & p2p3
#-----------------------------------------------------------
NODES_ID = ['obj_g1w1', 'obj_g2w2', 'obj_g3w3']
CUR_NODE_ID = 'obj_g3w3'

#-----------------------------------------------------------
# Number of Bytes received in a single communication
#-----------------------------------------------------------
# Linux OS:
    # TCP Receive Buffer Size (Min Default Max) in Bytes
        # $ cat /proc/sys/net/ipv4/tcp_rmem
        # $ sudo sysctl net ipv4.tcp_rmem
    # TCP Send Buffer Size (Min Default Max) in Bytes
        # $ cat /proc/sys/net/ipv4/tcp_wmem
        # $ sudo sysctl net ipv4.tcp_wmem
RECV_BYTES = 1048576 # Bytes

#-----------------------------------------------------------
# Step-3 & Step-7 Read-Trigger Filenames
#-----------------------------------------------------------
TRIGGERS = ['step3triggerw3g3.txt','step6triggerw3f3.txt']

#-----------------------------------------------------------
# Object names for Step-3 & Step-6 of DC-OMP_TA
#-----------------------------------------------------------
MAT_FILE_NAME3 = ["obj_g3w3", "obj_mat_all"]
MAT_FILE_NAME6 = ["obj_f3w3", "obj_mat_fs"]

#-----------------------------------------------------------
# Names of Pertinent DC-OMP_TA Steps
#-----------------------------------------------------------
TA_STEPS = ['step3', "step6"]

#-----------------------------------------------------------
# --------------------------END--------------------------
#-----------------------------------------------------------
```

4.7 p2p3.py

- This is a client/peer python program that runs on PC$_3$

```
#-----------------------------------------------------------
# Project_name: DC-OMP-TA Communications @ Step-3 & Step-6
# ------------------Peer/Client----------------------------
#-----------------------------------------------------------
#!/usr/bin/env python3
"""
Created on Saturday September 25 21:43:09 2021
@author: alem fitwi
"""

#-----------------------------------------------------------
# Import Necessary Libraries and Packages
#-----------------------------------------------------------
import os
import sys
import glob
from sys import path
import numpy as np
import socket
from scipy.io import loadmat, savemat
from pickle import dumps, loads
from _thread import *

#-----------------------------------------------------------
# Import All Necessary Constants, config files and Paths
#-----------------------------------------------------------
# Set the main absolute path (excluding file name) here!
COM_PATH = "/home/alem/Desktop/ntk/DC_OMP_TA_Project/"
path.append(os.path.join(COM_PATH,'config_mdl_p2p3/'))

# Import all user-defined parameters and configurations
from dcomptaconfigp2p3 import HEADER_SIZE,IO_PATH,\
    SERV_PORT,SERV_IP, NODES_ID, RECV_BYTES,\
    CUR_NODE_ID, TRIGGERS, MAT_FILE_NAME3, \
    MAT_FILE_NAME6, TA_STEPS, CONFIG_PATH

#-----------------------------------------------------------
# Read .mat file from IO_PATH, to be sent to remote peers
#-----------------------------------------------------------
def read_mat(IO_PATH, mat_name):
    """Reads .mat MATLAB Objects to disk"""
    filename = mat_name + ".mat"
    try:
        path1 = os.path.join(
              IO_PATH, filename)
        mat = loadmat(path1)
    except:
        return False
    else:
        return mat

#-----------------------------------------------------------
# Write .mat files fetched from remote peers to IO_PATH
#-----------------------------------------------------------
def save_mat(IO_PATH, mat_name, matobject):
    """Saves .mat MATLAB Objects to disk"""
    filename = mat_name + ".mat"
    try:
        path1 = os.path.join(
                    IO_PATH, filename)
        savemat(path1, matobject)
    except:
        return False
    else:
        return True

#-----------------------------------------------------------
# Serialize Objects using pickle.dumps
#-----------------------------------------------------------
def serialize(obj):
    """Pickles objects into byte-series"""
    try:
        ser = dumps(obj)
    except:
        return False
    else:
        return ser

#-----------------------------------------------------------
# Deserialize Objects using pickle.loads
#-----------------------------------------------------------
def deserialize(obj):
    """unpickles objects into objects"""
    try:
        deser = loads(obj)
    except:
        return False
    else:
        return deser
```

```python
#-----------------------------------------------------------
# Receive data from another peer
#-----------------------------------------------------------
def recv_obj(conn, RECV_BYTES, HEADER_SIZE):
    """Receives objects from other peers"""
    full_msg = b''
    while True:
        pyobj = conn.recv(RECV_BYTES)
        msg_len = int(pyobj[:HEADER_SIZE])
        full_msg += pyobj
        if len(full_msg) - HEADER_SIZE == msg_len:
            break
    return deserialize(full_msg[HEADER_SIZE:])

#-----------------------------------------------------------
# Send data to all other peers --> ['obj_g2w2', 'obj_g3w3']
#-----------------------------------------------------------
def send_obj(conn, obj, CUR_NODE_ID, step):
    """Sends pickled objects to other peers"""
    idd = CUR_NODE_ID + step
    obj1 = serialize(obj)
    obj1 = bytes(f"{idd}", "utf-8")+obj1
    obj3 =  bytes(f"{len(obj1):<{HEADER_SIZE}}",
                  "utf-8")+obj1
    conn.sendall(obj3)

#-----------------------------------------------------------
# Check steps-3&6 .mat read-trigger in IO_PATH Continually!
#-----------------------------------------------------------
def isTrigger(path2, TRIGGERS):
    """Checks step-3 & 6 Triggers"""
    fstep3 = os.path.join(path2,TRIGGERS[0])
    fstep6 = os.path.join(path2,TRIGGERS[1])
    bstep3 = os.path.exists(fstep3)
    bstep6 = os.path.exists(fstep6)
    return bstep3, bstep6

#-----------------------------------------------------------
# A Function that removes a file from disk
#-----------------------------------------------------------
def remove_file(path2, filename):
    """Removes files/objects from Disk"""
    try:
        path1 = os.path.join(path2, filename)
        file = glob.glob(path1)
        os.remove(file[0])
    except:
        print(sys.exc_info()[0])

#---------------------------------------------------------
# The Main Function where every other function is run!
#---------------------------------------------------------
if __name__ == '__main__':
    """Calls and executes all other functions"""

    rn = 1 # Rounds counter
    while True:
        # Reset Counter
        if rn > 65535:
            rn =1
    #------------------------------------------------------
        print('******************************************')
        print('Waiting For A Trigger From MATLAB ...')
        print('---------------------------------------')
        bstep3, bstep6 = isTrigger(IO_PATH, TRIGGERS)
        print(f"At Round {rn}")
        print('******************************************')
    #------------------------------------------------------
        if bstep3 == True:
            print('**************************************')
            disp_msg0 = f"Processing Step-3 of The "
            print(disp_msg0+f"DC_OMP_TA, @ Round {rn} ...")
            ClientSocket = socket.socket(socket.AF_INET,
                                  socket.SOCK_STREAM)
            print('**************************************')

    #------------------------------------------------------
            obj_g2w2 = read_mat(IO_PATH,MAT_FILE_NAME3[0])
            if isinstance(obj_g2w2, bool) == True:
                print('**********************************')
                print("No obj_g2w2.mat Object To Be Sent!")
                print('**********************************')
                continue
            else:
                print('**********************************')
                print('Local Object obj_g2w2 Read...')
                print('----------------------------------')
                print(f"obj_g2w2 = {obj_g2w2}")
                print('**********************************')
    #------------------------------------------------------
```

```
        try:
            ClientSocket.connect((SERV_IP, SERV_PORT))
        except socket.error as e:
            print(str(e))
#-----------------------------------------------------
        print('**************************************')
        disp_msg1 = "Waiting To Send Local Mat Object"
        print(disp_msg1 + " To Other Peers ...")
        print('--------------------------------------')
        send_obj(ClientSocket, obj_g2w2, CUR_NODE_ID,
                        TA_STEPS[0])
        print('--------------------------------------')
        print('obj_g2w2.mat Object Sent To Peers...')
        print(obj_g2w2)
        print('**************************************')
#-----------------------------------------------------
        print('**************************************')
        disp_msg3 = "Waiting To Receive Objects "
        print(disp_msg3 + 'From Other Peers...')
        print('--------------------------------------')
        obj_mat_all = recv_obj(ClientSocket,RECV_BYTES,
                                          HEADER_SIZE)
        disp_msg3 = "obj_mat_all.mat Received From "
        print(disp_msg3 + ' Other Peers ...')
        print('--------------------------------------')
        print(obj_mat_all)
        print('**************************************')

#-----------------------------------------------------
        print('**************************************')
        disp_msg4 = "Saving obj_mat_all.mat To "
        print(disp_msg4 + 'MATLAB Work Space...')
        print('--------------------------------------')
        save_mat(IO_PATH, MAT_FILE_NAME3[1],
                                           obj_mat_all)
        print("obj_mat_all Was Successfully Saved!")
        print('**************************************')
#-----------------------------------------------------
        print('**************************************')
        disp_msg5 = "Removing Triggers From MATLAB"
        print(disp_msg5+' Work Space...')
        print('--------------------------------------')
        remove_file(IO_PATH, TRIGGERS[0])
        remove_file(IO_PATH, MAT_FILE_NAME3[0]+".mat")
        eras = "Triggers/Inputs Were Successfully"
        print(eras + " Removed From Work Space!")
        print('**************************************')
#------------------------------------------------------
        print('**************************************')
        dp = f" @ Round {rn}"
        print(f"End of Step-3 Of The DC_OMP_TA"+dp)
        print('--------------------------------------')

#------------------------------------------------------
    elif bstep6 == True and bstep3 == False:
        print('**************************************')
        disp0 = f"Processing Step-6 Of The "
        print(disp0+f"DC_OMP_TA, @ Round {rn} ...")
        ClientSocket = socket.socket(socket.AF_INET,
                                  socket.SOCK_STREAM)
        print('**************************************')
#------------------------------------------------------
        obj_f2w2 = read_mat(IO_PATH,MAT_FILE_NAME6[0])
        if isinstance(obj_f2w2, bool) == True:
            print('**********************************')
            print("No obj_f2w2.mat Object To Be Sent!")
            print('**********************************')
            continue
        else:
            print('**********************************')
            print('Local Object obj_f2w2 Read...')
            print('----------------------------------')
            print(f"obj_f2w2 = {obj_f2w2}")
            print('**********************************')

#------------------------------------------------------
        try:
            ClientSocket.connect((SERV_IP, SERV_PORT))
        except socket.error as e:
            print(str(e))
#------------------------------------------------------
        print('**************************************')
        print('Waiting To Send Mat Object To Peers...')
        print('--------------------------------------')
        send_obj(ClientSocket, obj_f2w2, CUR_NODE_ID,
                                TA_STEPS[1])
        print('--------------------------------------')
        print('obj_f2w2.mat Object Sent To Peers...')
        print(obj_f2w2)
        print('--------------------------------------')
        print('**************************************')
    #-------------------------------------------------------
        print('**************************************')
        disp_msg6 = "Waiting To Receive Objects "
        print(disp_msg6 + 'From Other Peers...')
        print('--------------------------------------')
        obj_mat_fs = recv_obj(ClientSocket, RECV_BYTES,
                                          HEADER_SIZE)
        print('--------------------------------------')
        print('obj_mat_fs Received From Other Peers!')
        print(obj_mat_fs)
        print('**************************************')

    #-------------------------------------------------------
        print('**************************************')
        disp_msg7 = "Saving obj_mat_fs.mat To MATLAB"
        print(disp_msg7 + ' Work Space...')
        print('**************************************')
        save_mat(IO_PATH,MAT_FILE_NAME6[1],obj_mat_fs)
    #-------------------------------------------------------
        print('**************************************')
        disp_msg8 = "Removing Triggers From MATLAB "
        print(disp_msg8 + 'Work Space ...')
        print('--------------------------------------')
        remove_file(IO_PATH, TRIGGERS[1])
        remove_file(IO_PATH, MAT_FILE_NAME6[0]+".mat")
        tps =  "Triggers/Inputs were successfully "
        print(tps + "Removed From Work Space!")
        print('**************************************')

    #-------------------------------------------------------
        print('**************************************')
        dp2 = f" @ Round {rn}"
        print(f"End of Step-6 Of The DC_OMP_TA"+dp2)
        print('**************************************')
    #-------------------------------------------------------
      else:
        pass
      rn+=True
    #-------------------------------------------------------
    ClientSocket.close()

#-----------------------------------------------------------
# End of p2p2.py program, which acts like a peer/client!
#-----------------------------------------------------------
```

5. MATLAB Functions for Interfacing

When the MATLAB CODE of each nodes reach at either step 3 or step 6 of the DC-OMP-TA, it sends a trigger to the corresponding python code to initiate the exchange of MATLAB objects, namely sorted scores (g_l) and their corresponding set of indices (w_l).

5.1 send_and_recvp31.m

£ This is a MATLAB function to be placed @ step3 of DC-OMP-TA running on PC_1

```
function send_and_recvp31(g1, w1, d_ID1, trig_fname31)
%-------------------------------------------------------
% Variables Assignments
local_obj_name31 = strcat(d_ID1, ".mat");

% Set a flag for handling a waiting loop
flag31 = 'False';
%-------------------------------------------------------
% Delete stale local and global MATLAB objects
delete(local_obj_name31);
delete obj_mat_all.mat;
%clear g1 w1;

%-------------------------------------------------------
% Save scores (g) and corresponding indices (w) as .mat
save(local_obj_name31, "g1", "w1");

%-------------------------------------------------------
% Send a trigger to python p2p1.py that object is ready
fid = fopen(trig_fname31,'wt');
disp("---------------------------------------------")
fprintf(fid, 'communication is triggered!');
disp("---------------------------------------------")
fclose(fid);

%-------------------------------------------------------
% Wait until [(g1, w1)] & [(g2, w2), (g3, w3)] are sent
% to other peers and received from other peers,
% respectively.
disp("---------------------------------------------")
disp("Waiting for updates from other peers...");
disp("---------------------------------------------")

while 1
    try
        %clear all;
        load obj_mat_all.mat;%obj_g1w1.mat;
        flag31 = 'True';
    catch
        flag31 = 'False';
        disp("---------------------------------------------")
        disp("Still waiting for updates ...");
        disp("---------------------------------------------")
    end

    if strcmp(flag31,'True')
        break;
    else
        continue;
    end

end
disp("--------------------------------------------")
disp("Scores & indices received from other peers...");
disp("--------------------------------------------")
%-------------------------------------------------------
% Remove trigger flag from disk
delete(trig_fname31);
%-------------------------------------------------------
end
%-------------------------------------------------------
```

£ Call this function at the end of step-3 of DC-OMP-TA on PC$_1$:

```
%-------------------------------------------------------
% @ Step-3, on PC1:
function send_and_recvp31(g1, w1, "obj_g1w1", "step3triggerw1g1.txt")
%-------------------------------------------------------
```

5.2 send_and_recvp32.m

£ This is a MATLAB function to be placed @ step3 of DC-OMP-TA running on PC$_2$

```
function send_and_recvp32(g2, w2, d_ID2, trig_fname32)
%-------------------------------------------------------
% Variables Assignments
local_obj_name32 = strcat(d_ID2, ".mat")

% Set a flag for handling a waiting loop
flag32 = 'False';
%-------------------------------------------------------
% Delete stale local and global MATLAB objects
delete(local_obj_name32);
delete obj_mat_all.mat;
%clear g2 w2;

%-------------------------------------------------------
% Save scores (g) and corresponding indices (w) as .mat
save(local_obj_name32, "g2", "w2");

%-------------------------------------------------------
% Send a trigger to python p2p2.py that object is ready
fid = fopen(trig_fname32,'wt');
fprintf (fid, 'Step3: communication is triggered!');
fclose(fid);
```

```
%------------------------------------------------------
% Wait until [(g2, w2)] & [(g1, w1), (g3, w3)] are sent
% to other peers and received from other peers,
% respectively.
disp ("Waiting for updates from other peers...");

while 1
    try
        %clear all;
        load obj_mat_all.mat;
        flag32 = 'True';
    catch
        flag32 = 'False';
        disp("------------------------------------------------");
        disp ("Still waiting for updates ...");
        disp("------------------------------------------------");
    end

    if strcmp(flag32,'True')
        break;
    else
        continue;
    end

end
disp("------------------------------------------------");
disp("Scores & indices received from other peers...");
disp("------------------------------------------------");

%------------------------------------------------------
% Remove trigger flag from disk
delete(trig_fname32);
%------------------------------------------------------
end
%------------------------------------------------------
```

£ Call this function at the end of step-3 of DC-OMP-TA on PC_2:

```
%------------------------------------------------------
% @ Step-3, on PC2:
function send_and_recvp32(g2, w2, "obj_g2w2", "step3triggerw2g2.txt")
%------------------------------------------------------
```

5.3 send_and_recvp33.m

£ This is a MATLAB function to be placed @ step3 of DC-OMP-TA running on PC_3

```
function send_and_recvp33(g3, w3, d_ID3, trig_fname33)
%------------------------------------------------------
% Variables Assignments
local_obj_name33 = strcat (d_ID3, ".mat");

% Set a flag for handling a waiting loop
```

```
flag33 = 'False';
%-------------------------------------------------------
% Delete stale local and global MATLAB objects
delete(local_obj_name33);
delete obj_mat_all.mat;
%clear g3 w3;

%-------------------------------------------------------
% Save scores (g) and corresponding indices (w) as .mat
save (local_obj_name33, "g3", "w3");

%-------------------------------------------------------
% Send a trigger to python p2p3.py that object is ready
fid = fopen(trig_fname33,'wt');
disp("--------------------------------------------")
fprintf (fid, 'Step3: communication is triggered!');
disp("'Step3: communication is triggered!");
disp("--------------------------------------------")
fclose(fid);

%-------------------------------------------------------
% Wait until [(g3, w3)] & [(g2, w2), (g1, w1)] are sent
% to other peers and received from other peers,
% respectively.
disp("--------------------------------------------")
disp ("Waiting for updates from other peers...");
disp("--------------------------------------------")

while 1
    try
        %clear all;
        load obj_mat_all.mat;
        flag33 = 'True';
        disp(flag33)
    catch
        flag33 = 'False';
        disp("--------------------------------------------")
        disp ("Still waiting for updates ...");
        disp("--------------------------------------------")
    end

    if strcmp(flag33,'True')
        break;
    else
        continue;
    end

end
disp("--------------------------------------------")
disp ("Scores & indices received from other peers...");
disp("--------------------------------------------")

%-------------------------------------------------------
% Remove trigger flag from disk
delete(trig_fname33);
```

```
%-------------------------------------------------------
end
%-------------------------------------------------------
```

£ Call this function at the end of step-3 of DC-OMP-TA on PC_3:

```
%-------------------------------------------------------
% @ Step-3, on PC3:
function send_and_recvp33(g3, w3, "obj_g3w3", "step3triggerw3g3.txt")
%-------------------------------------------------------
```

5.4 send_and_recvp61.m

£ This is a MATLAB function to be placed @ step6 of DC-OMP-TA running on PC_1

```
function send_and_recvp61(ff1, ww1, fn61, trig_fname61)
%-------------------------------------------------------
% Variables Assignments
local_obj_name61 = strcat (fn61, ".mat");

% Set a flag for handling a waiting loop
flag_main61 = 'False';
%-------------------------------------------------------
% Delete stale local and global MATLAB objects
delete(local_obj_name61);
delete obj_mat_fs.mat; %Set of updates
%clear ff1 ww1;

%-------------------------------------------------------
% Save scores (g) and corresponding indices (w) as .mat
% If no updates, please set ff to empty!
if isempty(ww1)
    flag = 'False';
    save (local_obj_name61, "ff1", "ww1", "flag");
else
    flag = 'True';
    save(local_obj_name61, "ff1", "ww1", "flag");
end
%-------------------------------------------------------
% Send a trigger to python p2p1.py that object is ready
fid = fopen(trig_fname61,'wt');
disp("---------------------------------------------")
fprintf (fid, 'Step6: communication is triggered!');
disp("---------------------------------------------")
fclose(fid);

%-------------------------------------------------------
% Wait until [(f1, w1)] & [(f2, w2), (f3, w3)] are sent
% to other peers and received from other peers,
% respectively.
disp("---------------------------------------------")
disp("Waiting for updates from other peers...");
disp("---------------------------------------------")
```

```
while 1
    try
        %clear all;
        load obj_mat_fs.mat;
        flag_main61 = 'True';
    catch
        flag_main61 = 'False';
        disp("--------------------------------------------")
        disp("Still waiting for updates ...");
        disp("--------------------------------------------")
    end

    if strcmp(flag_main61,'True')
        break;
    else
        continue;
    end

end
disp("--------------------------------------------")
disp ("Scores updates between peers is complete ...");
disp("--------------------------------------------")
%-------------------------------------------------------
% Remove trigger flag from disk
delete(trig_fname61);
%-------------------------------------------------------
end
%-------------------------------------------------------
```

£ Call this function at the end of step-6 of DC-OMP-TA on PC_1:

```
%-------------------------------------------------------
% @ Step-6, on PC1:
function send_and_recvp61(ff1, ww1, "obj_f1w1", "step6triggerw1f1.txt")
%-------------------------------------------------------
```

5.5 send_and_recvp62.m

£ This is a MATLAB function to be placed @ step6 of DC-OMP-TA running on PC_2

```
function send_and_recvp62(ff2, ww2, fn62, trig_fname62)
%-------------------------------------------------------
% Variables Assignments
local_obj_name62 = strcat (fn62, ".mat")

% Set a flag for handling a waiting loop
flag_main62 = 'False';
%-------------------------------------------------------
% Delete stale local and global MATLAB objects
delete(local_obj_name62);
delete obj_mat_fs.mat; %Dictionary of updates
%clear ff2 ww2;
```

```
%-----------------------------------------------------
% Save scores (g) and corresponding indices (w) as .mat
% If no updates, please set ff to empty!
if isempty(ww2)
    flag = 'False';
    save(local_obj_name62, "ff2", "ww2", "flag");
else
    flag = 'True';
    save(local_obj_name62, "ff2", "ww2", "flag");
end
%-----------------------------------------------------
% Send a trigger to python p2p2.py that object is ready
fid = fopen(trig_fname62,'wt');
fprintf (fid, 'Step6: communication is triggered!');
fclose(fid);

%-----------------------------------------------------
% Wait until [(f1, w1)] & [(f2, w2), (f3, w3)] are sent
% to other peers and received from other peers,
% respectively.
disp("-----------------------------------------------");
disp("Waiting for updates from other peers...");
disp("-----------------------------------------------");

while 1
    try
        %clear all;
        load obj_mat_fs.mat;
        flag_main62 = 'True';
    catch
        flag_main62 = 'False';
        disp("-----------------------------------------------");
        disp ("Still waiting for updates ...");
        disp("-----------------------------------------------");
    end

    if strcmp(flag_main62,'True')
        break;
    else
        continue;
    end

end
disp("-----------------------------------------------");
disp ("Scores updates between peers is complete ...");
disp("-----------------------------------------------");
%-----------------------------------------------------
% Remove trigger flag from disk
delete(trig_fname62);
%-----------------------------------------------------
end
%-----------------------------------------------------
```

£ Call this function at the end of step-6 of DC-OMP-TA on PC$_2$:

```
%-------------------------------------------------------
% @ Step-6, on PC2:
function send_and_recvp62(ff2, ww2, "obj_f2w2", "step6triggerw2f2.txt")
%-------------------------------------------------------
```

5.6 send_and_recvp63.m

£ This is a MATLAB function to be placed @ step6 of DC-OMP-TA running on PC$_3$

```
function send_and_recvp63(ff3, ww3, fn63, trig_fname63)
%-------------------------------------------------------
% Variables Assignments
local_obj_name63 = strcat (fn63, ".mat");

% Set a flag for handling a waiting loop
flag_main63 = 'False';
%-------------------------------------------------------
% Delete stale local and global MATLAB objects
delete(local_obj_name63);
delete obj_mat_fs.mat; %Dictionary of updates
%clear ff3 ww3;

%-------------------------------------------------------
% Save scores (g) and corresponding indices (w) as .mat
% If no updates, please set ff to empty!
if isempty(ww3);
    flag = 'False';
    save(local_obj_name63, "ff3", "ww3", "flag");
else
    flag = 'True';
    save(local_obj_name63, "ff3", "ww3", "flag");
end
%-------------------------------------------------------
% Send a trigger to python p2p3.py that object is ready
fid = fopen(trig_fname63,'wt');
disp("--------------------------------------------")
fprintf (fid, 'Step6: communication is triggered!');
disp('Step6: communication is triggered!')
disp("--------------------------------------------")
fclose(fid);

%-------------------------------------------------------
% Wait until [(f3, w3)] & [(f2, w2), (f1, w1)] are sent
% to other peers and received from other peers,
% respectively.
disp("--------------------------------------------")
disp ("Waiting for updates from other peers...");
disp("--------------------------------------------")

while 1
    try
        %clear all;
        load obj_mat_fs.mat;
```

```
            flag_main63 = 'True';
        catch
            flag_main63 = 'False';
            disp("--------------------------------------------")
            disp ("Still waiting for updates ...");
            disp("--------------------------------------------")
        end

        if strcmp(flag_main63,'True')
            break;
        else
            continue;
        end

    end
    disp("--------------------------------------------")
    disp ("Scores updates between peers is complete ...");
    disp("--------------------------------------------")
    %-----------------------------------------------------
    % Remove trigger flag from disk
    delete(trig_fname63);
    %-----------------------------------------------------
    end
    %-----------------------------------------------------
```

£ Call this function at the end of step-6 of DC-OMP-TA on PC3:

```
%-----------------------------------------------------
%- @ Step-6, on PC3:
function send_and_recvp63(ff3, ww3, "obj_f3w3", "step6triggerw3f3.txt")
%-----------------------------------------------------
```

6. File Organization

This project's files and objects are organized and placed into two main folders namely "*DC_OMP_TA_Project*" and "MATLAB" based on whether the files and objects are germane to the python application or to the MATLAB function programs. The MATLAB folder refers to the default folder of the MATLAB software installed on the machine of interest. As shown in the table below, the "*DC_OMP_TA_Project*" folder permanently contains one of the three python programs, and a corresponding configuration file. The "MATLAB" folder should permanently contain two MATLAB functions (`send_and_recvp31.m, & send_and_recvp61.m`). However, when program control reaches at step-3 and step-6, the "MATLAB" folder will additionally store two temporary local objects (obj_g1w1.mat & obj_f1w1.mat), two global objects (obj_mat_all.mat & obj_mat_fs.mat), and triggers (step3triggerw1g1.txt and step6triggerw1f1.txt). The lifetime of these objects and triggers is between step-3 and step-6. They are all cleared at the end of their corresponding step and built again in the next round. For maximum clarity, the file organization at each PC is pictorially explained in what ensues subsection wise.

6.1 File Organization @PC-1

The files, objects or programs in PC-1 are briefly described in the table below and their organization is portrayed in Fig 5.

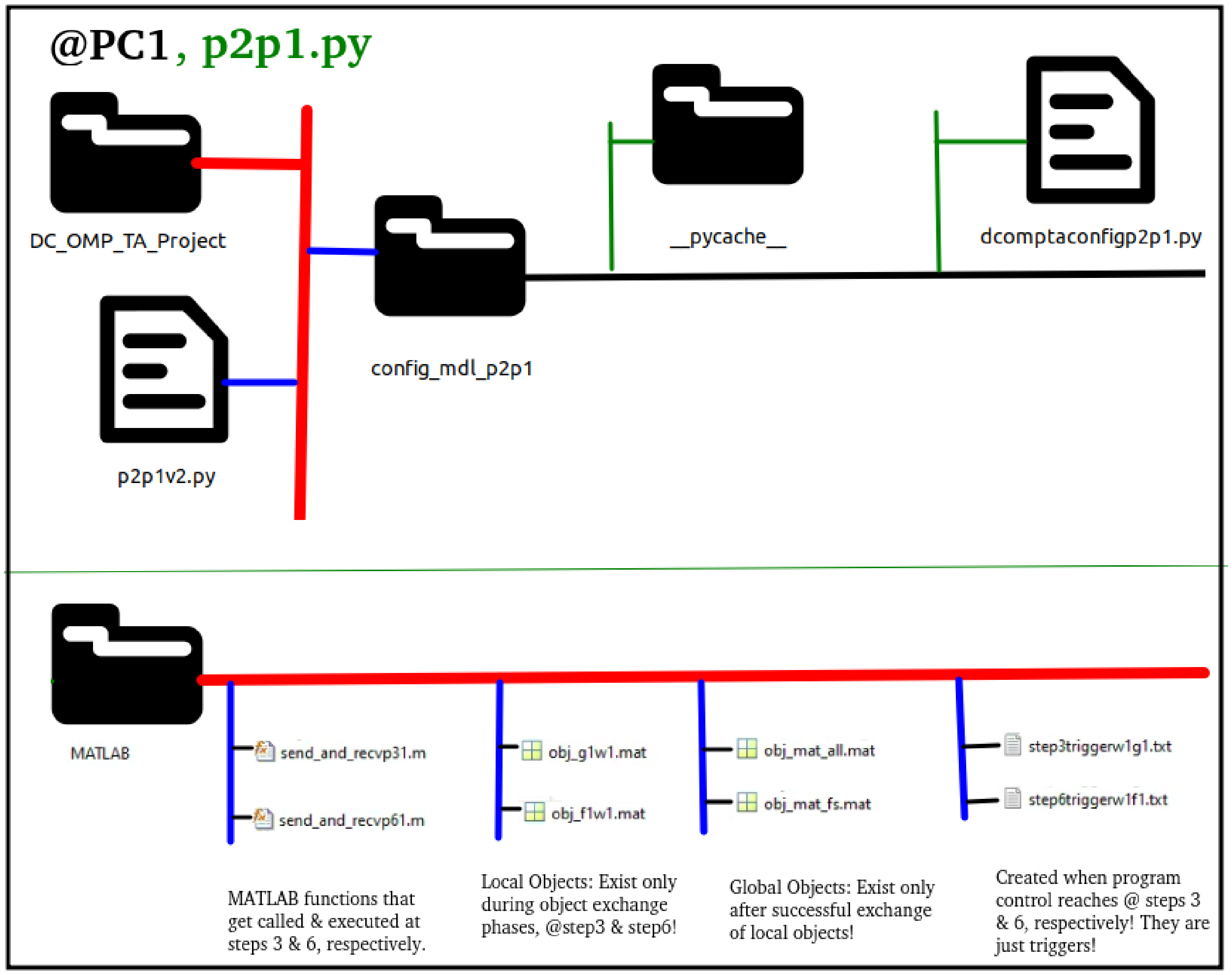

@PC1, p2p1.py
DC_OMP_TA_Project
config_mdl_p2p1
__pycache__
dcomptaconfigp2p1.py
p2p1v2.py
MATLAB
send_and_recvp31.m
send_and_recvp61.m
obj_g1w1.mat
obj_f1w1.mat
obj_mat_all.mat
obj_mat_fs.mat
step3triggerw1g1.txt
step6triggerw1f1.txt
MATLAB functions that get called & executed at steps 3 & 6, respectively.
Local Objects: Exist only during object exchange phases, @step3 & step6!
Global Objects: Exist only after successful exchange of local objects!
Created when program control reaches @ steps 3 & 6, respectively! They are just triggers!

Fig 5 File Structure/Organization @ PC-1

File/Program/Object	***Description***
p2p1v2.py	A python that helps for establishing communication with other peers
dcomptaconfigp2p1.py	It defines all configurations and constants employed in p2p1v2.py program
send_and_recvp31.m	A MATLAB function that sends read-trigger signal to p2p1v2.py @step3
send_and_recvp61.m	A MATLAB function that sends read-trigger signal to p2p1v2.py @step6
obj_g1w1.mat	Dictionary of local objects to be transmitted to other peers created by function-31
obj_f1w1.mat	Dictionary of local objects to be transmitted to other peers created by function-61
obj_mat_all.mat	Dictionary of all local objects received from other peers @ step-3
obj_mat_fs.mat	Dictionary of all local objects received from other peers @ step-6

6.2 File Organization @PC-2

File/Program/Object	***Description***

p2p2v2.py	A python that helps for establishing communication with other peers
dcomptaconfigp2p2.py	It defines all configurations and constants employed in p2p2v2.py program
send_and_recvp32.m	A MATLAB function that sends read-trigger signal to p2p2v2.py @step3
send_and_recvp62.m	A MATLAB function that sends read-trigger signal to p2p2v2.py @step6
obj_g2w2.mat	Dictionary of local objects to be transmitted to other peers created by function-32
obj_f2w2.mat	Dictionary of local objects to be transmitted to other peers created by function-62
obj_mat_all.mat	Dictionary of all local objects received from other peers @ step-3
obj_mat_fs.mat	Dictionary of all local objects received from other peers @ step-6

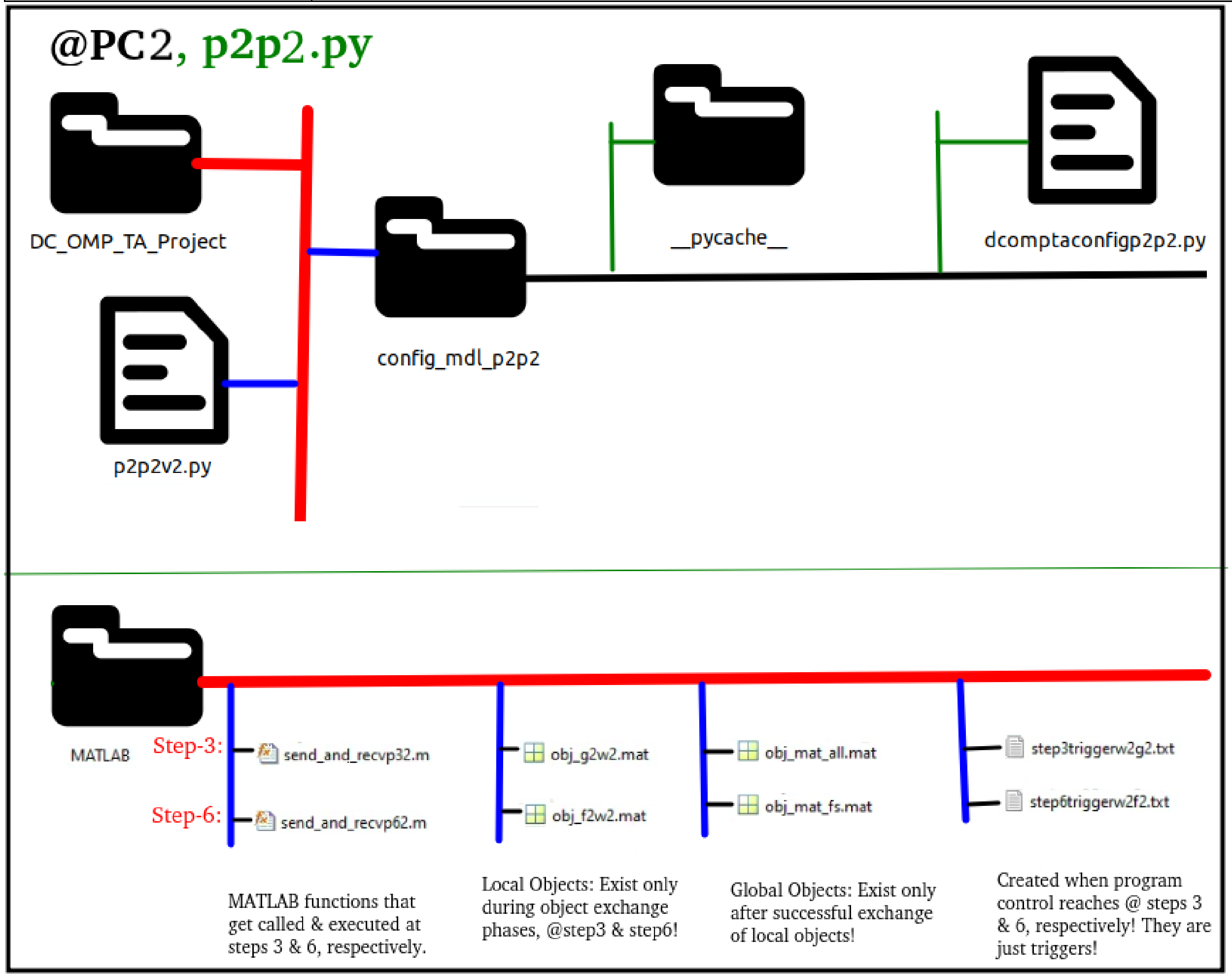

@PC2, p2p2.py
DC_OMP_TA_Project
__pycache__
dcomptaconfigp2p2.py
config_mdl_p2p2
p2p2v2.py
MATLAB
Step-3:
Step-6:
send_and_recvp32.m
send_and_recvp62.m
obj_g2w2.mat
obj_f2w2.mat
obj_mat_all.mat
obj_mat_fs.mat
step3triggerw2g2.txt
step6triggerw2f2.txt
MATLAB functions that get called & executed at steps 3 & 6, respectively.
Local Objects: Exist only during object exchange phases, @step3 & step6!
Global Objects: Exist only after successful exchange of local objects!
Created when program control reaches @ steps 3 & 6, respectively! They are just triggers!

Fig 6 File Structure/Organization @ PC-2

6.3 File Organization @PC-3

File/Program/Object	*Description*
p2p3v2.py	A python that helps for establishing communication with other peers
dcomptaconfigp2p3.py	It defines all configurations and constants employed in p2p3v2.py program

send_and_recvp33.m	A MATLAB function that sends read-trigger signal to p2p3v2.py @step3
send_and_recvp63.m	A MATLAB function that sends read-trigger signal to p2p3v2.py @step6
obj_g3w3.mat	Dictionary of local objects to be transmitted to other peers created by function-33
obj_f3w3.mat	Dictionary of local objects to be transmitted to other peers created by function-63
obj_mat_all.mat	Dictionary of all local objects received from other peers @ step-3
obj_mat_fs.mat	Dictionary of all local objects received from other peers @ step-6

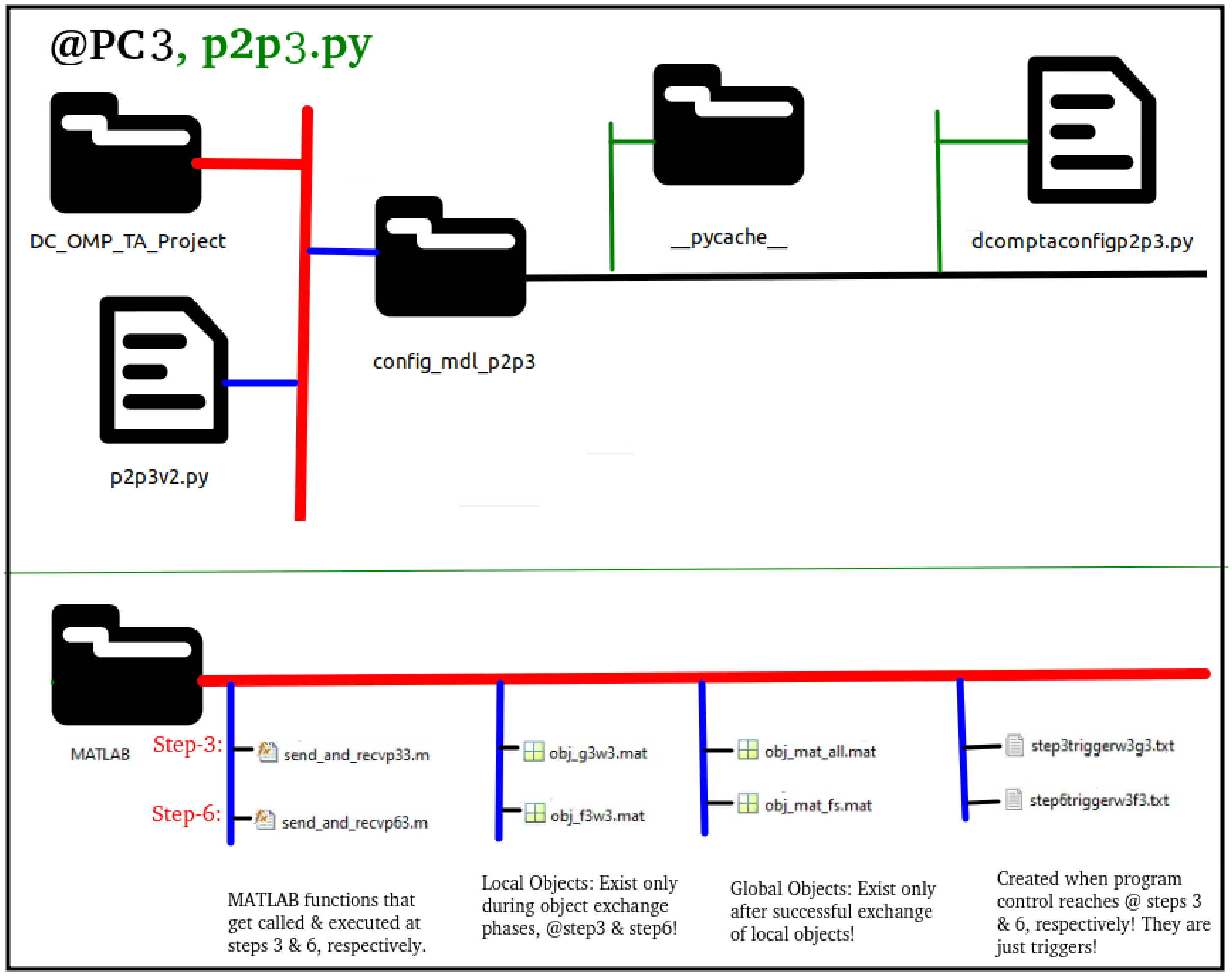

@PC3, p2p3.py
DC_OMP_TA_Project
config_mdl_p2p3
__pycache__
dcomptaconfigp2p3.py
p2p3v2.py
MATLAB
Step-3:
Step-6:
send_and_recvp33.m
send_and_recvp63.m
obj_g3w3.mat
obj_f3w3.mat
obj_mat_all.mat
obj_mat_fs.mat
step3triggerw3g3.txt
step6triggerw3f3.txt
MATLAB functions that get called & executed at steps 3 & 6, respectively.
Local Objects: Exist only during object exchange phases, @step3 & step6!
Global Objects: Exist only after successful exchange of local objects!
Created when program control reaches @ steps 3 & 6, respectively! They are just triggers!

Fig 7 File Structure/Organization @ PC-3

7. Test Results

According to our testing plan, a series of two tests are to be carried out. The first test is done on a single computer by running all peers on separate command lines and using three separate folders to emulate the roles of MATLAB software running on three networked machines. This test was already conducted successfully. The second test was carried out on three independent machines connected to each other by means of a LAN, CAN, or WAN network.

7.1 Test on a Single Machine using Multiple CMDs

To test the developed apps on a single machine, all three of the python codes (p2p1.py, p2p2.py, & p2p3.py) along with their corresponding configuration files (dcomptaconfigp2p1.py, dcomptaconfigp2p2.py, & dcomptaconfigp2p3.py) and three empty folders (matlab1, matlab2, & matlab3) for the emulation of MATLAB running on three machines are organized in the way portrayed in Fig 8. As can be seen from the diagram in Fig 8, folders 'matlab1', 'matlab2', & 'matlab3' are empty. They initially contain no objects or triggers!

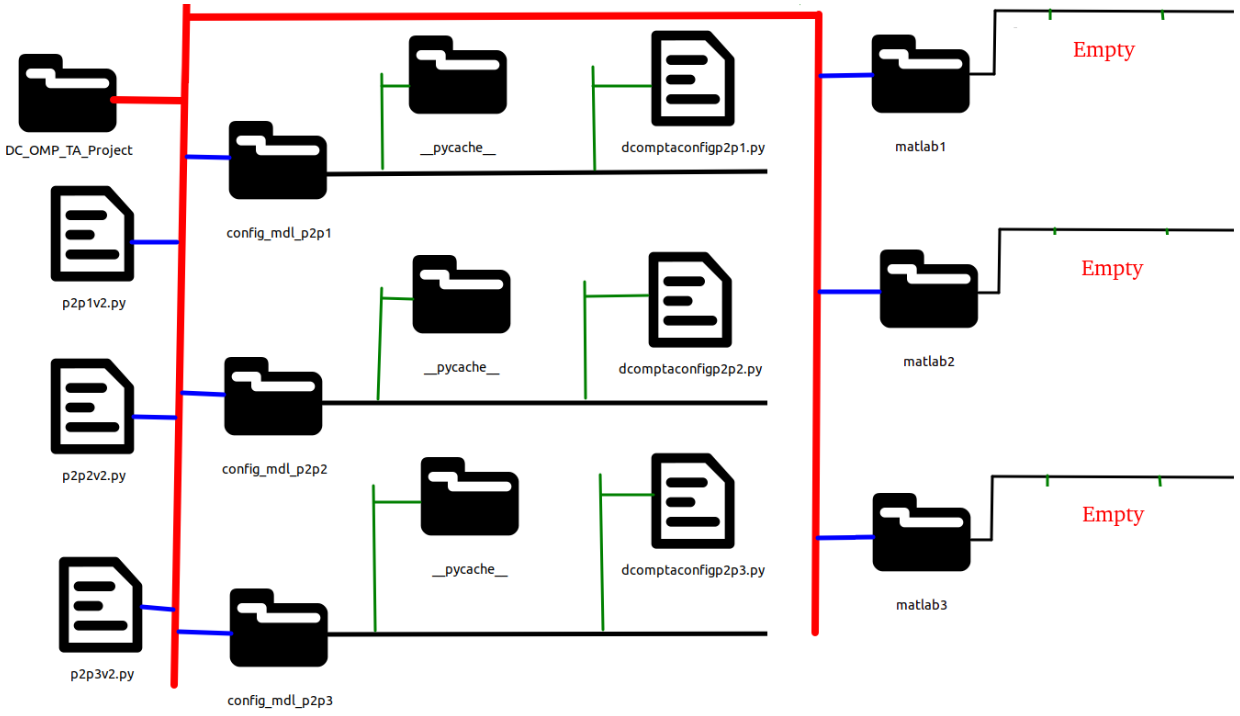
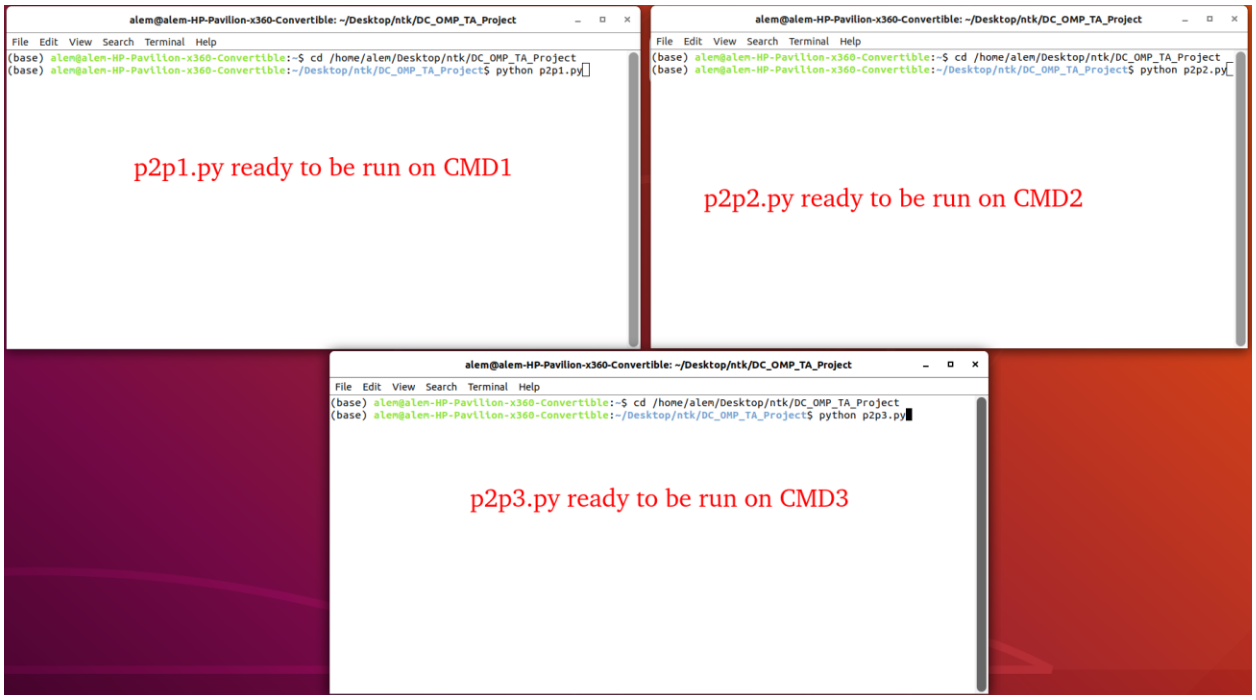


Fig 8 Organization of All Programs and files for testing on a single machine.

To begin the testing, we opened three CLIs as illustrated in Fig 9 and made the three python apps ready for running. The intention behind opening three CLIs is to emulate the actual implementation on three networked machines. Next, all the apps were executed and the peers were set into running mode as shown in Fig 10. To save resources, the peers are designed to make communications with one another only when they have some objects or updates to exchange with one another, as clearly described in Algorithm 1. The connection phase is facilitated by the peer designed to play a server role. At step-3, all peers must send their scores (this is based on the nature of the DC_OMP_TA algorithm). As a result, the process will remain in blocked mode until all peers send their sorted scores to their neighbors and the exchange process is successfully completed. At step-6, only peers that have new scores or updates are supposed to transmit. Peers with no updates send an empty mat object with its flag set to "False"; otherwise, set to "True". Hence, no need to block the process!


alem@alem-HP-Pavilion-x360-Convertible: ~/Desktop/ntk/DC_OMP_TA_Project
File Edit View Search Terminal Help

```
(base) alem@alem-HP-Pavilion-x360-Convertible:~$ cd /home/alem/Desktop/ntk/DC_OMP_TA_Project
(base) alem@alem-HP-Pavilion-x360-Convertible:~/Desktop/ntk/DC_OMP_TA_Project$ python p2p1.py
```

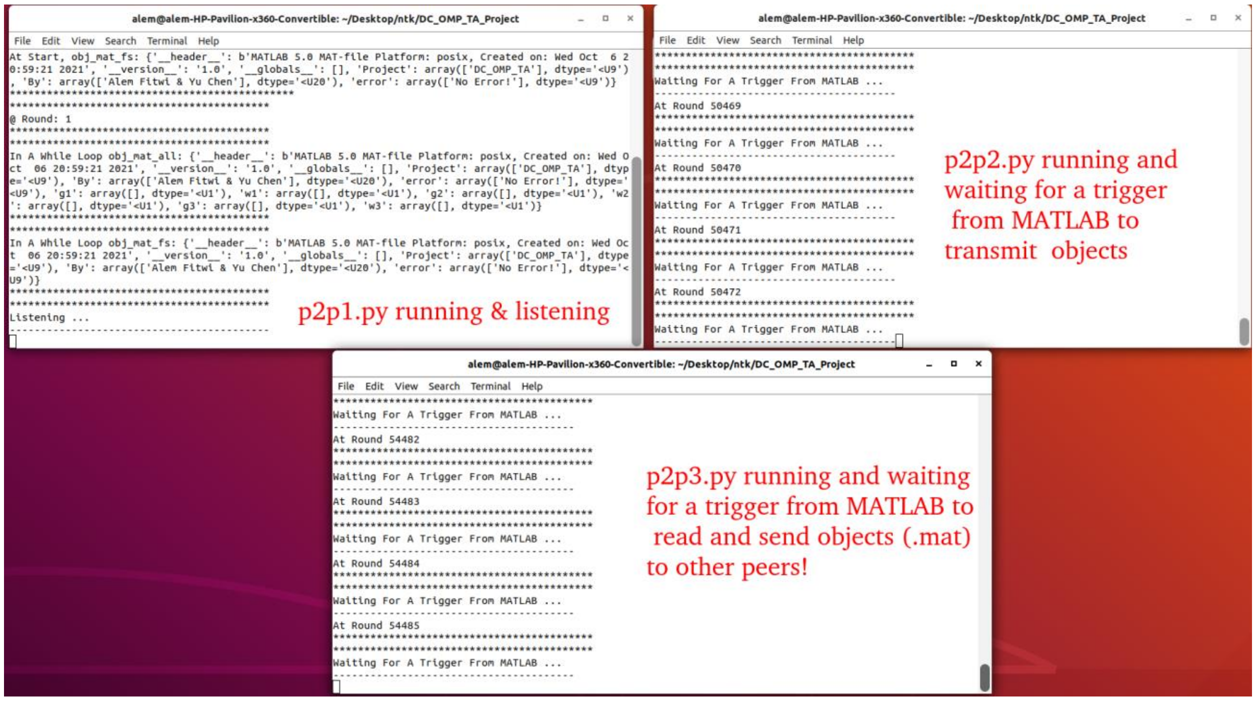
p2p1.py ready to be run on CMD1

```
(base) alem@alem-HP-Pavilion-x360-Convertible:~$ cd /home/alem/Desktop/ntk/DC_OMP_TA_Project
(base) alem@alem-HP-Pavilion-x360-Convertible:~/Desktop/ntk/DC_OMP_TA_Project$ python p2p2.py
```

p2p2.py ready to be run on CMD2

```
(base) alem@alem-HP-Pavilion-x360-Convertible:~$ cd /home/alem/Desktop/ntk/DC_OMP_TA_Project
(base) alem@alem-HP-Pavilion-x360-Convertible:~/Desktop/ntk/DC_OMP_TA_Project$ python p2p3.py
```

Fig 9 Three opened CLI Screens that emulate 3 machines, the python apps are ready to be run.


alem@alem-HP-Pavilion-x360-Convertible: ~/Desktop/ntk/DC_OMP_TA_Project
File Edit View Search Terminal Help

```
At Start, obj_mat_fs: {'__header__': b'MATLAB 5.0 MAT-file Platform: posix, Created on: Wed Oct  6 2
0:59:21 2021', '__version__': '1.0', '__globals__': [], 'Project': array(['DC_OMP_TA'], dtype='<U9')
, 'By': array(['Alem Fitwi & Yu Chen'], dtype='<U20'), 'error': array(['No Error!'], dtype='<U9')}
**********************************************
******************************************
@ Round: 1
******************************************
******************************************
In A While Loop obj_mat_all: {'__header__': b'MATLAB 5.0 MAT-file Platform: posix, Created on: Wed O
ct  06 20:59:21 2021', '__version__': '1.0', '__globals__': [], 'Project': array(['DC_OMP_TA'], dtyp
e='<U9'), 'By': array(['Alem Fitwi & Yu Chen'], dtype='<U20'), 'error': array(['No Error!'], dtype='
<U9'), 'g1': array([], dtype='<U1'), 'w1': array([], dtype='<U1'), 'g2': array([], dtype='<U1'), 'w2
': array([], dtype='<U1'), 'g3': array([], dtype='<U1'), 'w3': array([], dtype='<U1')}
******************************************
******************************************
In A While Loop obj_mat_fs: {'__header__': b'MATLAB 5.0 MAT-file Platform: posix, Created on: Wed Oc
t  06 20:59:21 2021', '__version__': '1.0', '__globals__': [], 'Project': array(['DC_OMP_TA'], dtype
='<U9'), 'By': array(['Alem Fitwi & Yu Chen'], dtype='<U20'), 'error': array(['No Error!'], dtype='<
U9')}
******************************************
******************************************
Listening ...
------------------------------------------
```

```
******************************************
******************************************
Waiting For A Trigger From MATLAB ...
----------------------------------------
At Round 50469
******************************************
******************************************
Waiting For A Trigger From MATLAB ...
----------------------------------------
At Round 50470
******************************************
******************************************
Waiting For A Trigger From MATLAB ...
----------------------------------------
At Round 50471
******************************************
******************************************
Waiting For A Trigger From MATLAB ...
----------------------------------------
At Round 50472
******************************************
******************************************
Waiting For A Trigger From MATLAB ...
----------------------------------------
```

p2p2.py running and waiting for a trigger from MATLAB to transmit objects

```
******************************************
Waiting For A Trigger From MATLAB ...
----------------------------------------
At Round 54482
******************************************
******************************************
Waiting For A Trigger From MATLAB ...
----------------------------------------
At Round 54483
******************************************
******************************************
Waiting For A Trigger From MATLAB ...
----------------------------------------
At Round 54484
******************************************
******************************************
Waiting For A Trigger From MATLAB ...
----------------------------------------
At Round 54485
******************************************
******************************************
Waiting For A Trigger From MATLAB ...
----------------------------------------
```

p2p3.py running and waiting for a trigger from MATLAB to read and send objects (.mat) to other peers!

Fig 10 The three-python peer/server apps are running, and listening for triggers from MATLAB.

After the objects and triggers have been placed on the MATLAB folders manually (as depicted in Fig 10) while the python apps are running to emulate what the MATLAB would do in an actual distributed processing on three networked machines, local objects have been exchanged between peers. As a result, all peers have global view of one another just before the end of step-6 as shown in Fig 12. Then, the objects can be easily loaded to the main MATLAB code at any of the PCs for further processing! The local objects & triggers are cleared after the completion of pertinent steps.

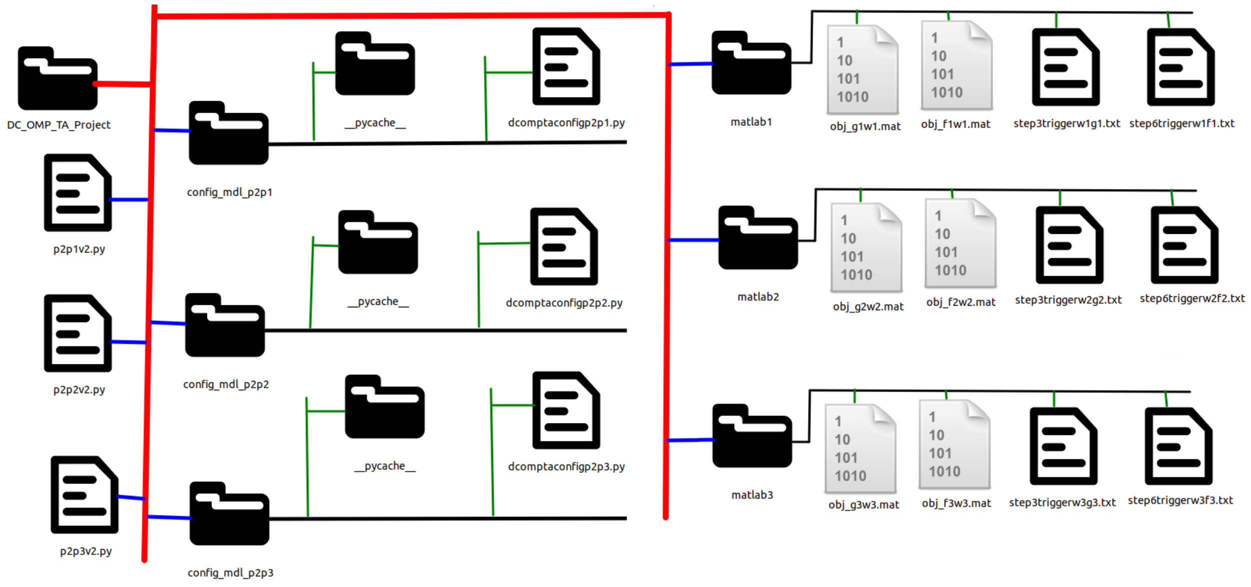


Fig 11 Objects and triggers are manually added to the three MATLAB folders

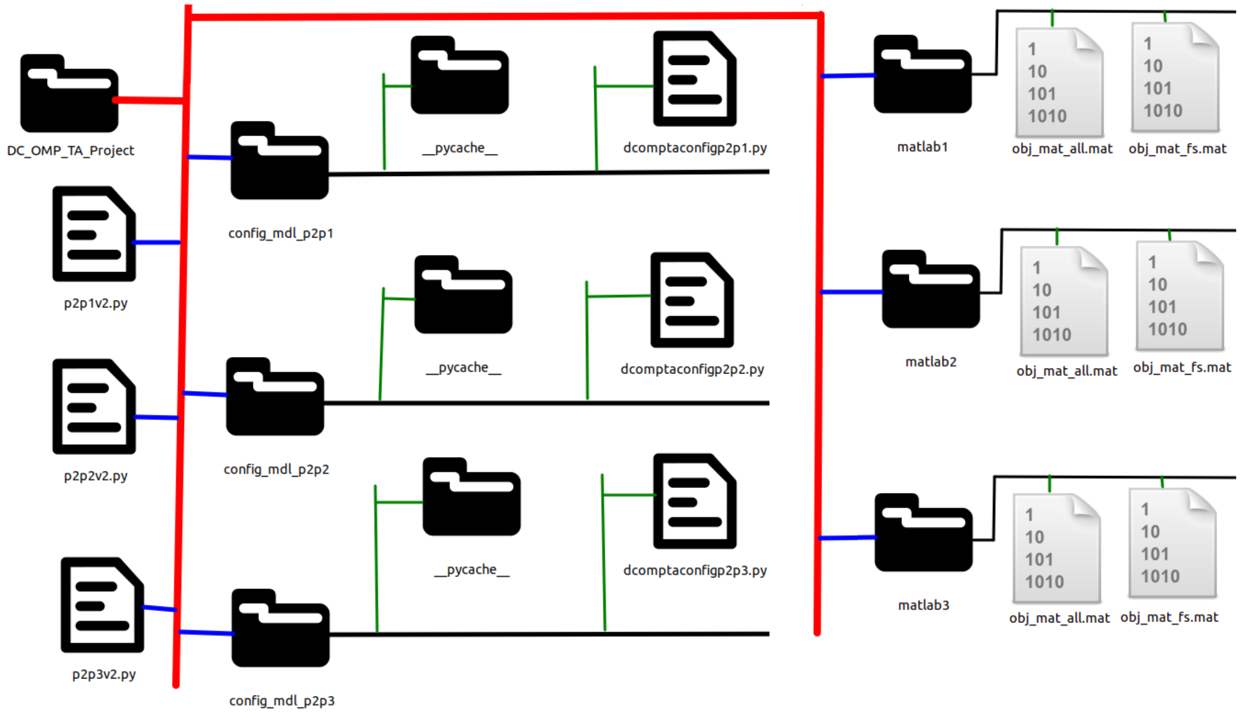


Fig 12 All PCs have now global view of one another, just at the end of step 6 before reset.

Once step-3 and step-6 are completed, the apps will return into waiting states. They will remain in listening mode until another set of triggers are made by the MATLAB programs.

7.2 Test Based-on Three Separate Networked Virtual Machines

For this test, three virtual machines (VM), namely pc1, pc2, and pc3 with static IP addresses 128.226.80.61, 128.226.80.62, 128.226.80.63, respectively, were created on vSphere. The file structure or organization in each VM is clearly portrayed in Fig 13.

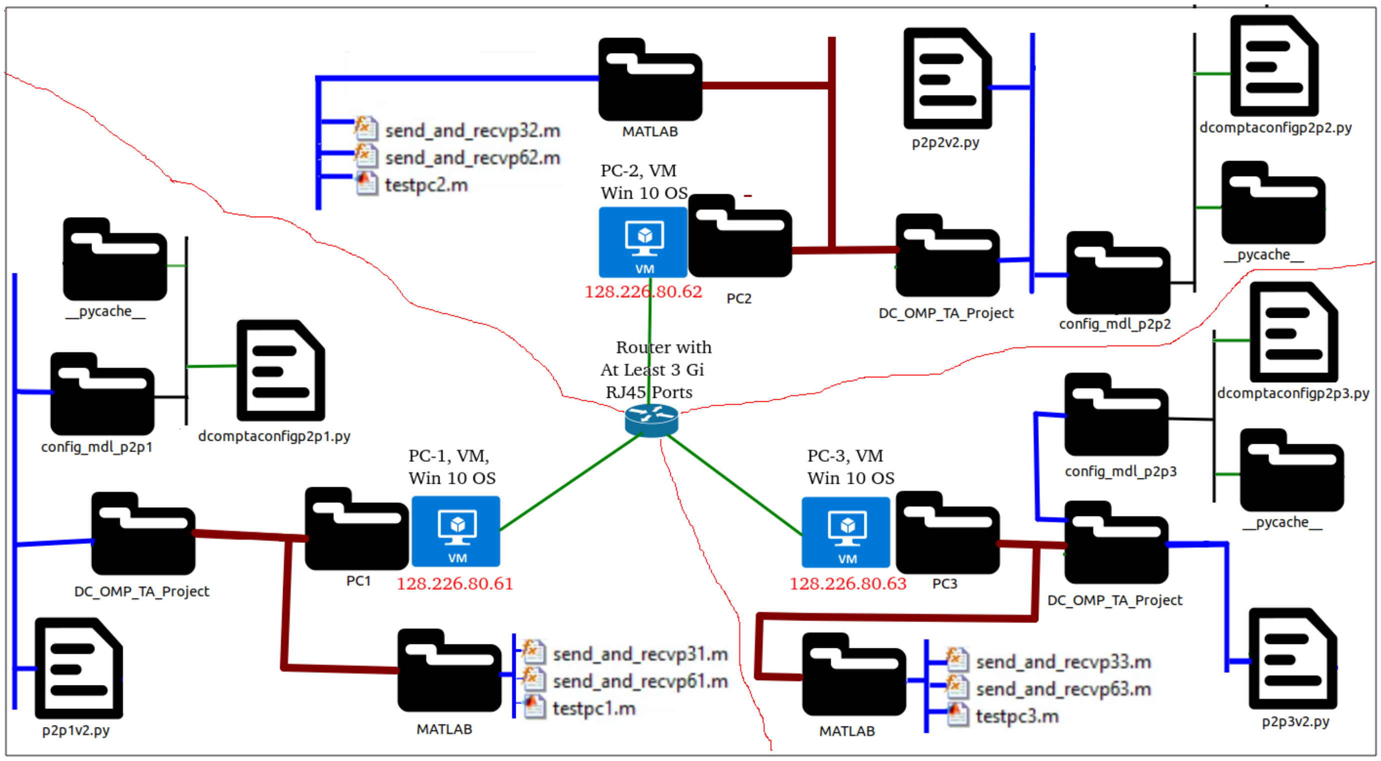


Fig 13 Setup for Testing Based-on Three Virtual Machines, namely PC$_1$, PC$_2$, and PC$_3$.

To emulate the actual MATLAB program of the DC_OMP_TA, we created dummy MATLAB code meant only for testing the proposed communication algorithm on distributed machines. This test code is run on three of the VMs in parallel and it calls the functions developed for steps 3 and 6 to create local objects and triggers whenever the control transfer reaches at the target steps. Time delays are introduced to simply simulate the parts of the DC_OMP_TA program before step-3, between step-3 and step-6, and after step-6. The test MATLAB program is named as testpc1.m, testpc2.m, and testpc3.m on PC-1, PC-2, and PC-3, respectively, as clearly illustrated in Fig 13.

The test was successful! Local objects were created by the MATLAB codes and the exchanging process was successfully carried out by the python peer/server apps. Finally, the global-view of objects was achieved!

Please observe the following steps:

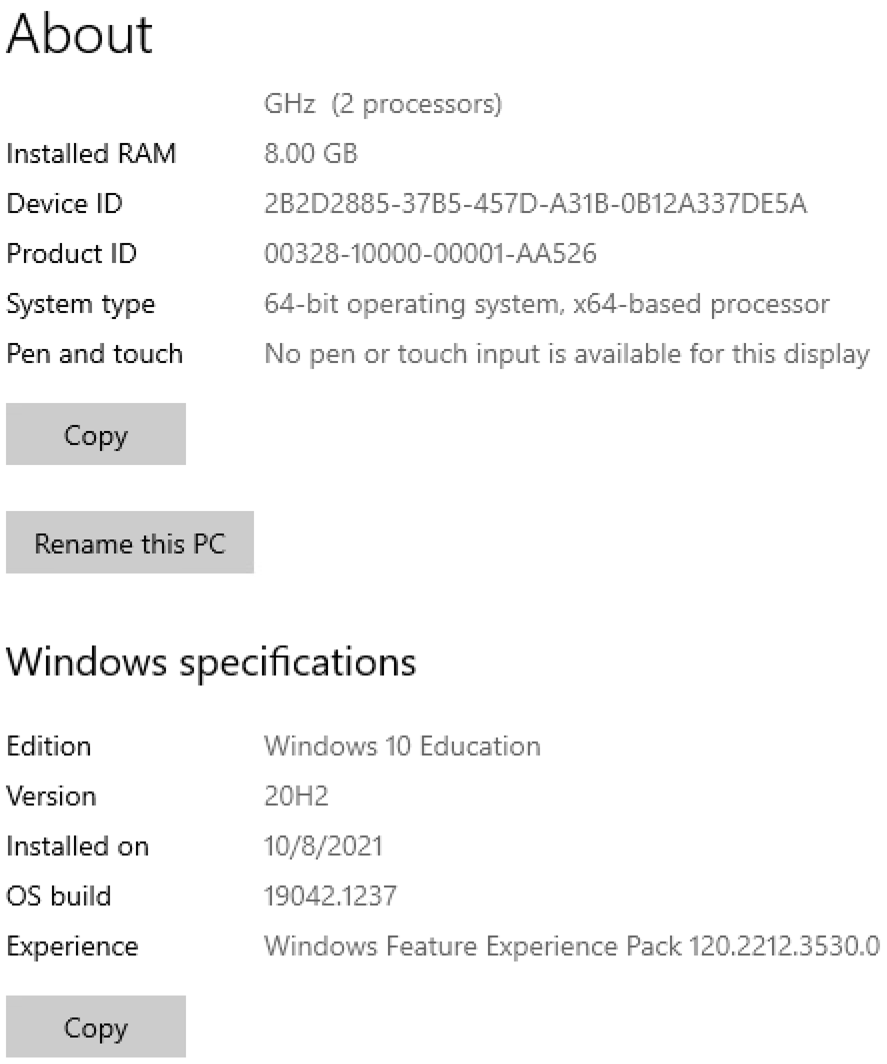
a) Place all required files in each pc as shown in Fig 13
b) Make all paths are set correctly in both the python and MATLAB programs
c) Correctly set the IP addresses on the configuration file of each python program.
d) Run python code p2p1v2.py on pc1 first for it is the coordinator; then, run p2p2v2.py and p2p3v2.py. The latter two can be run in any order.

e) Run your MATLAB codes that call the interface function on each machine. You can use our dummy MATLAB programs (testpc1.m, testpc2.m, and testpc3.m) for starters.
f) After successful runs, you should be able to see obj_mat_all.mat on the MATLAB folder of all computers involved.

8. Conclusions

£ The python programs (peer/server) have been successfully tested on a local machine (CPU: i5 2.5GHz 4 core, RAM: 4GB, OS: Ubuntu 18.04.5 LTS). They run correctly as desired and are able to establish the required communication between the three peers at a speed of light, without any further ado!

£ Testing on three separate networked virtual machines was also conducted. The specification of each VM is provided in Fig 14. Local objects were created by the MATLAB codes and the exchanging process was successfully carried out by the python peer/server apps. Finally, the global-view of objects was achieved in every pc involved!



About

	GHz (2 processors)
Installed RAM	8.00 GB
Device ID	2B2D2885-37B5-457D-A31B-0B12A337DE5A
Product ID	00328-10000-00001-AA526
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

Windows specifications

Edition	Windows 10 Education
Version	20H2
Installed on	10/8/2021
OS build	19042.1237
Experience	Windows Feature Experience Pack 120.2212.3530.0

Fig 14: Specification of the 3 VMs employed for testing

£ The performance of such communication is heavily dependent on the TCP/IP RECV BUFFER size. Hence, setting the right size of the TCP RECV Buffer on your machines is so vital during actual implementation. For instance, if you have 2MB of objects to be transmitted but your TCP_RECV_BUFFER size is 4096B, then this message will be broken into 512 chunks each having a size of 4096B and a peer will complete sending this data to other peers in 512 rounds of communications! But if you have a TCP RECEV BUFFER size >= 2MB, the objects are likely sent in a single round of communication.

Important Side Note:

€ In a number of WANs, the round-trip time (RTT) is dominated by the propagation delay. Long RTTs along and TCP buffer size have throughput implications.

€ Consider a 5 Gbps WAN with a 50-millisecond RTT. Assume that the TCP send and receive buffer size is set to 1Mbyte (1 MB = 1024^2 bytes = 1,048,576 bytes = 1,048,576 * 8 bits = 8, 388, 608 bits). With a bandwidth (Bw) of 5 Gbps, this number of bits is approximately transmitted in

$$T_{tx} = \frac{\#\,of\ Bits}{Bandwidth} = \frac{8{,}388{,}608\ \text{bits}}{5 * 2^{30} bits/s} = 0.0015625\ seconds$$

€ After 1.5 milliseconds, the content of the TCP send buffer will be completely sent. At this point, TCP must wait for the corresponding acknowledgements, which will only start arriving at t = 50 milliseconds. This means that the sender only uses 1.5/50 or 3% of the available bandwidth.

€ The remedy to the above problem lies in allowing the sender to continuously transmit segments until the corresponding acknowledgments arrive back. There shouldn't be a buffer limitation at the sender end; data to be sent should only be decided by the TCP Window Size. Note that the first acknowledgement arrives after an RTT. The number of bits that can be transmitted in an RTT period is given by the bandwidth of the channel in bits per second multiplied by the RTT. This quantity is referred to as the ***bandwidth-delay product (BDP)***. Hence, the **recv_buffer** size must be greater than or equal to the ***BDP***:

$$TCP\_BUFFER\_SIZE \geq BDP$$

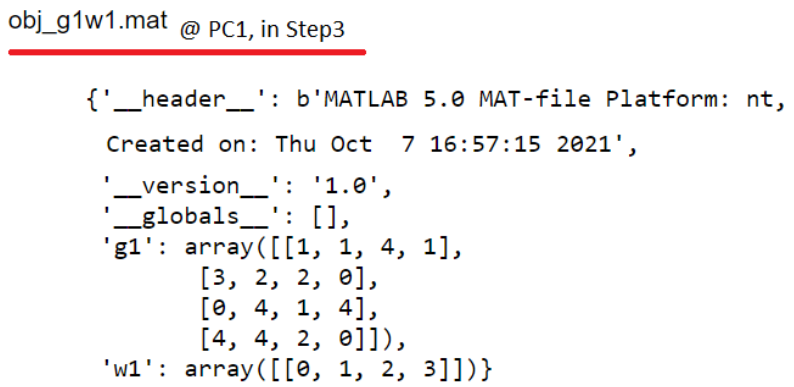
€ Use either of the following cmds on a CLI to check your machine's TCP Buffer sizes

```
# $ cat /proc/sys/net/ipv4/tcp_rmem
   4096      131072      6291456
# $ sudo sysctl net.ipv4.tcp_rmem
   4096      131072      6291456
```

£ You can transfer or receive whatever size of objects or data using the proposed communication architecture; however, the greater the size of data exchanged, the longer will be the transmission time, especially if you have smaller bandwidth. A smaller TCP_RECV_BUFFER size will also increase the propagation delay!

£ It is mandatory that whatever data or object to be sent to other peers must be in MAT object format; that is, in a dictionary of dictionaries. And the outer keys at every machine PC_1, PC_2, or PC_3 should always be [g1, w1], [g2, w2], & [g3, w3] for step-3 and [ff1, ww1], [ff2, ww2], & [ff3, ww3] for step-6, respectively. Whenever you add a new machine to the network, make sure to rename the important variables and constants accordingly! For the sake of clarity, MAT-files are binary MATLAB objects that store workspace variables. They are very easily created and accessed in any MATLAB program.

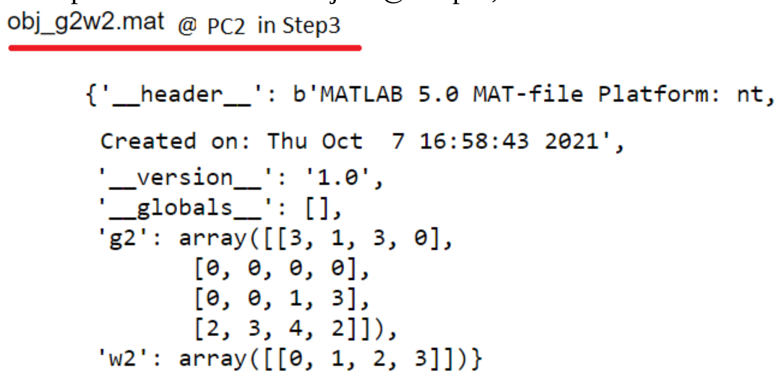
£ Example of Local MAT-Object @ Step-3, on PC1:



obj_g1w1.mat @ PC1, in Step3

```
{'__header__': b'MATLAB 5.0 MAT-file Platform: nt,
 Created on: Thu Oct  7 16:57:15 2021',
 '__version__': '1.0',
 '__globals__': [],
 'g1': array([[1, 1, 4, 1],
        [3, 2, 2, 0],
        [0, 4, 1, 4],
        [4, 4, 2, 0]]),
 'w1': array([[0, 1, 2, 3]])}
```

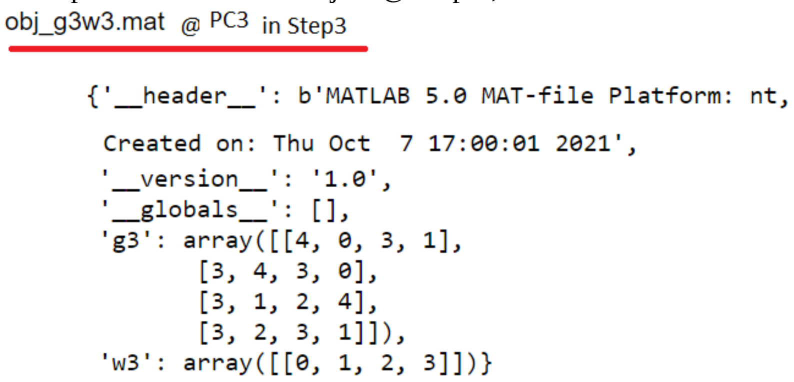
£ Example of Local MAT-Object @ Step-3, on PC2:



obj_g2w2.mat @ PC2 in Step3

```
{'__header__': b'MATLAB 5.0 MAT-file Platform: nt,
 Created on: Thu Oct  7 16:58:43 2021',
 '__version__': '1.0',
 '__globals__': [],
 'g2': array([[3, 1, 3, 0],
        [0, 0, 0, 0],
        [0, 0, 1, 3],
        [2, 3, 4, 2]]),
 'w2': array([[0, 1, 2, 3]])}
```

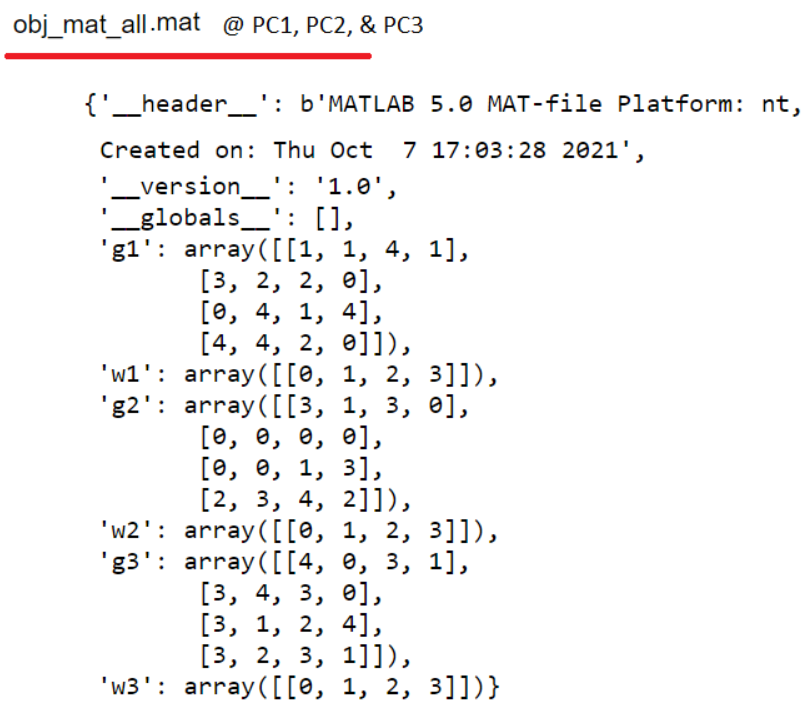
£ Example of Local MAT-Object @ Step-3, on PC3:



obj_g3w3.mat @ PC3 in Step3

```
{'__header__': b'MATLAB 5.0 MAT-file Platform: nt,
 Created on: Thu Oct  7 17:00:01 2021',
 '__version__': '1.0',
 '__globals__': [],
 'g3': array([[4, 0, 3, 1],
        [3, 4, 3, 0],
        [3, 1, 2, 4],
        [3, 2, 3, 1]]),
 'w3': array([[0, 1, 2, 3]])}
```

£ Example of Global MAT-Object @ Step-3, shared on PC1, PC2, & PC3. The PCs achieve a global view of one another ensuing the successful exchange of local MAT-objects (obj_g1w1.mat, obj_g2w2.mat, & obj_g3w3.mat).

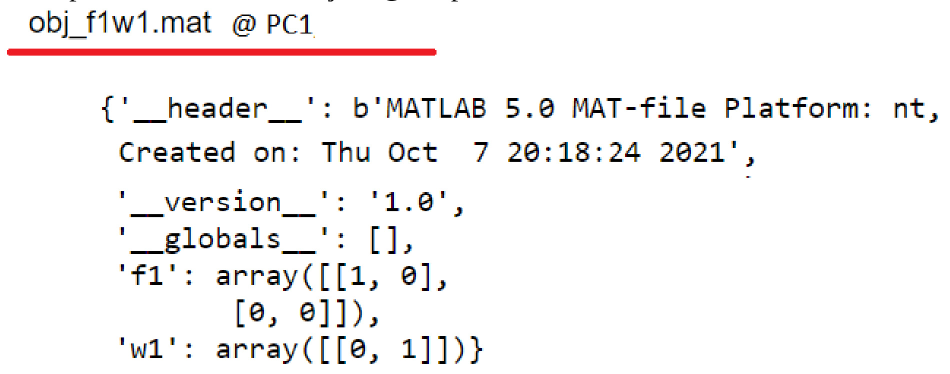

obj_mat_all.mat @ PC1, PC2, & PC3

```
{'__header__': b'MATLAB 5.0 MAT-file Platform: nt,
 Created on: Thu Oct  7 17:03:28 2021',
 '__version__': '1.0',
 '__globals__': [],
 'g1': array([[1, 1, 4, 1],
        [3, 2, 2, 0],
        [0, 4, 1, 4],
        [4, 4, 2, 0]]),
 'w1': array([[0, 1, 2, 3]]),
 'g2': array([[3, 1, 3, 0],
        [0, 0, 0, 0],
        [0, 0, 1, 3],
        [2, 3, 4, 2]]),
 'w2': array([[0, 1, 2, 3]]),
 'g3': array([[4, 0, 3, 1],
        [3, 4, 3, 0],
        [3, 1, 2, 4],
        [3, 2, 3, 1]]),
 'w3': array([[0, 1, 2, 3]])}
```

£ Eventually, you can easily load (load obj_mat_all.mat) the relevant scores (g) and their respective indices (w) to your MATLAB program to do whatever you want on them next to step-3.

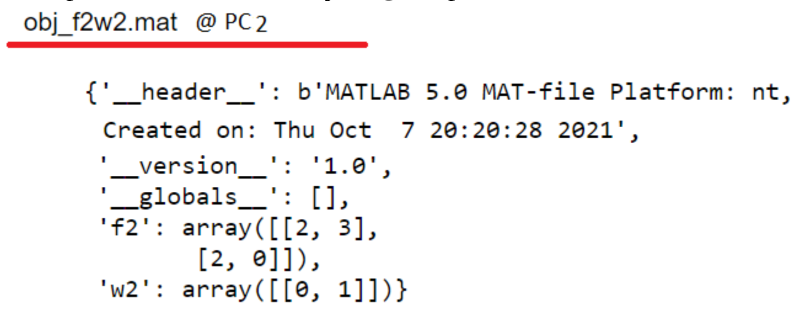
£ Likewise, we can demonstrate the local and global objects generated by step-6 of the DC_OMP_TA. As a brief recall, obj_f1w1.mat, obj_f2w2.mat, and obj_f3w3.mat are the local object files of step-6 at machines PC1, PC2, and PC3, respectively. The object with global view, created after the successful exchange of local objects, is obj_mat_fs.mat.

£ Example of Local MAT-Object @ Step-6, on PC1:


obj_f1w1.mat @ PC1

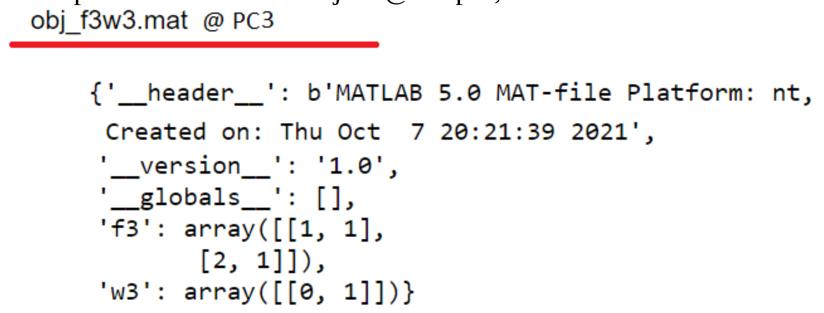
```
{'__header__': b'MATLAB 5.0 MAT-file Platform: nt,
 Created on: Thu Oct  7 20:18:24 2021',
 '__version__': '1.0',
 '__globals__': [],
 'f1': array([[1, 0],
        [0, 0]]),
 'w1': array([[0, 1]])}
```

£ Example of Local MAT-Object @ Step-6, on PC2:


obj_f2w2.mat @ PC2

```
{'__header__': b'MATLAB 5.0 MAT-file Platform: nt,
 Created on: Thu Oct  7 20:20:28 2021',
 '__version__': '1.0',
 '__globals__': [],
 'f2': array([[2, 3],
        [2, 0]]),
 'w2': array([[0, 1]])}
```

£ Example of Local MAT-Object @ Step-6, on PC3:



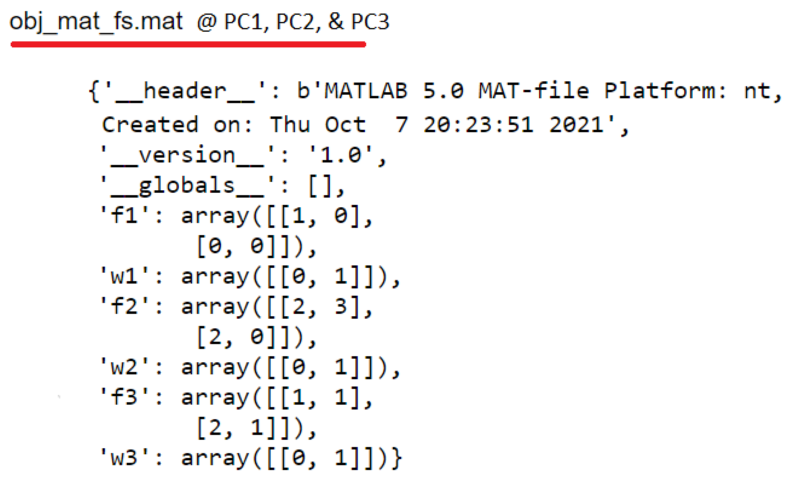
obj_f3w3.mat @ PC3

```
{'__header__': b'MATLAB 5.0 MAT-file Platform: nt,
 Created on: Thu Oct  7 20:21:39 2021',
 '__version__': '1.0',
 '__globals__': [],
 'f3': array([[1, 1],
        [2, 1]]),
 'w3': array([[0, 1]])}
```

£ ***NB***: unlike @ step-3 where every nodemust exchange sorted scores with their neighbors, at step-6 only those peers that have newly created scores or indices send objects to their neighbors. The table below illustrates what happen at each PC during the execution of step-6.

PC	**Objects**	**Status @ Step6**	**flag**	**Remark**
PC_1	f1, w1	f1, w1 update	“True”	Updates sent
PC_2	f2, w1	No update	“False”	“False” sent
PC_3	f3, w3	f1, w1 update	“True”	Updates sent

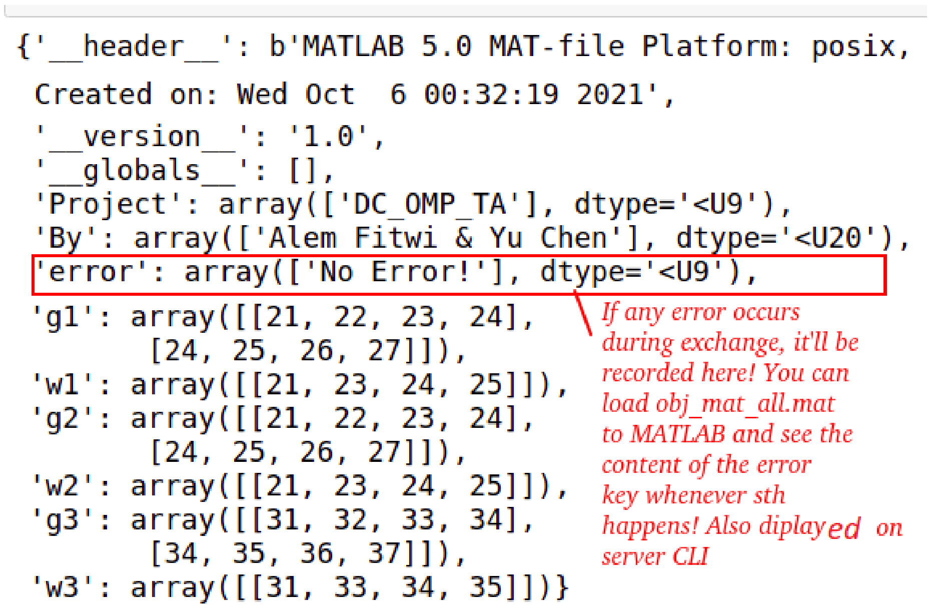
£ At end of the step-6 object updates exchange, all PCs or nodes will have the same global views updates made on each PC, as portrayed in the figure below. Then, the main MATLAB program can easily load (load obj_mat_fs.mat) the relevant updates of scores (f) and their respective indices (w) and act up on them starting at the step next to step-3.



obj_mat_fs.mat @ PC1, PC2, & PC3

```
{'__header__': b'MATLAB 5.0 MAT-file Platform: nt,
 Created on: Thu Oct  7 20:23:51 2021',
 '__version__': '1.0',
 '__globals__': [],
 'f1': array([[1, 0],
        [0, 0]]),
 'w1': array([[0, 1]]),
 'f2': array([[2, 3],
        [2, 0]]),
 'w2': array([[0, 1]]),
 'f3': array([[1, 1],
        [2, 1]]),
 'w3': array([[0, 1]])}
```

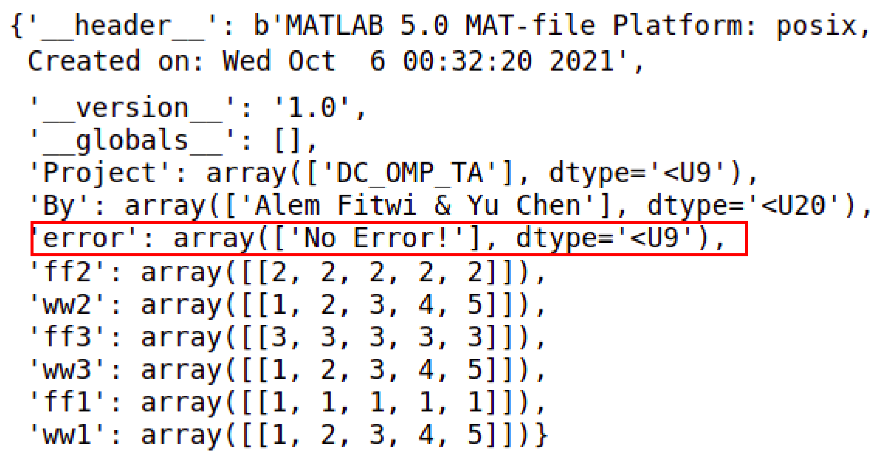
£ Global object of step-3 (obj_mat_all.mat) with all details included. All the machines exchange their local objects at step-3, a final object containing all the local objects is created on all of them as shown below. If any kind of error occurs in the process of creating a global view of the objects, it is caught and recorded as part of the obj_mat_all.mat file with a key value “error”, as clearly illustrated below.



```
{'__header__': b'MATLAB 5.0 MAT-file Platform: posix,
 Created on: Wed Oct  6 00:32:19 2021',
 '__version__': '1.0',
 '__globals__': [],
 'Project': array(['DC_OMP_TA'], dtype='<U9'),
 'By': array(['Alem Fitwi & Yu Chen'], dtype='<U20'),
 'error': array(['No Error!'], dtype='<U9'),
 'g1': array([[21, 22, 23, 24],
        [24, 25, 26, 27]]),
 'w1': array([[21, 23, 24, 25]]),
 'g2': array([[21, 22, 23, 24],
        [24, 25, 26, 27]]),
 'w2': array([[21, 23, 24, 25]]),
 'g3': array([[31, 32, 33, 34],
        [34, 35, 36, 37]]),
 'w3': array([[31, 33, 34, 35]])}
```

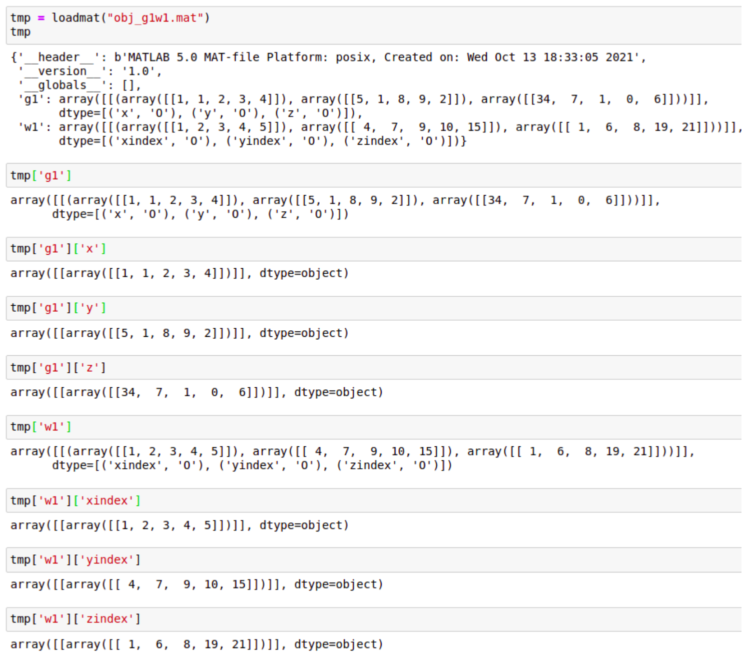
If any error occurs during exchange, it'll be recorded here! You can load obj_mat_all.mat to MATLAB and see the content of the error key whenever sth happens! Also diplayed on server CLI

£ Global object of step-3 (obj_mat_all.mat) with all details included. The error reporting approach is employed with the obj_mat_fs.mat global object of step-6, as portrayed below



```
{'__header__': b'MATLAB 5.0 MAT-file Platform: posix,
 Created on: Wed Oct  6 00:32:20 2021',
 '__version__': '1.0',
 '__globals__': [],
 'Project': array(['DC_OMP_TA'], dtype='<U9'),
 'By': array(['Alem Fitwi & Yu Chen'], dtype='<U20'),
 'error': array(['No Error!'], dtype='<U9'),
 'ff2': array([[2, 2, 2, 2, 2]]),
 'ww2': array([[1, 2, 3, 4, 5]]),
 'ff3': array([[3, 3, 3, 3, 3]]),
 'ww3': array([[1, 2, 3, 4, 5]]),
 'ff1': array([[1, 1, 1, 1, 1]]),
 'ww1': array([[1, 2, 3, 4, 5]])}
```

£ *Using this approach, you can transfer or receive any number of objects (the bandwidth is the upper limit). You should only make sure that there are only two outer keys (gl & wl or ff1 & ww1) in every MAT-File created locally. For example, if you want to transmit multiple objects (x, y, & z) form node-1, it should be done as portrayed in the figure below.*



```
tmp = loadmat("obj_g1w1.mat")
tmp
```

```
{'__header__': b'MATLAB 5.0 MAT-file Platform: posix, Created on: Wed Oct 13 18:33:05 2021',
 '__version__': '1.0',
 '__globals__': [],
 'g1': array([[(array([[1, 1, 2, 3, 4]]), array([[5, 1, 8, 9, 2]]), array([[34,  7,  1,  0,  6]]))]],
       dtype=[('x', 'O'), ('y', 'O'), ('z', 'O')]),
 'w1': array([[(array([[1, 2, 3, 4, 5]]), array([[ 4,  7,  9, 10, 15]]), array([[ 1,  6,  8, 19, 21]]))]],
       dtype=[('xindex', 'O'), ('yindex', 'O'), ('zindex', 'O')])}
```

```
tmp['g1']
```

```
array([[(array([[1, 1, 2, 3, 4]]), array([[5, 1, 8, 9, 2]]), array([[34,  7,  1,  0,  6]]))]],
      dtype=[('x', 'O'), ('y', 'O'), ('z', 'O')])
```

```
tmp['g1']['x']
```

```
array([[array([[1, 1, 2, 3, 4]])]], dtype=object)
```

```
tmp['g1']['y']
```

```
array([[array([[5, 1, 8, 9, 2]])]], dtype=object)
```

```
tmp['g1']['z']
```

```
array([[array([[34,  7,  1,  0,  6]])]], dtype=object)
```

```
tmp['w1']
```

```
array([[(array([[1, 2, 3, 4, 5]]), array([[ 4,  7,  9, 10, 15]]), array([[ 1,  6,  8, 19, 21]]))]],
      dtype=[('xindex', 'O'), ('yindex', 'O'), ('zindex', 'O')])
```

```
tmp['w1']['xindex']
```

```
array([[array([[1, 2, 3, 4, 5]])]], dtype=object)
```

```
tmp['w1']['yindex']
```

```
array([[array([[ 4,  7,  9, 10, 15]])]], dtype=object)
```

```
tmp['w1']['zindex']
```

```
array([[array([[ 1,  6,  8, 19, 21]])]], dtype=object)
```

£ *Make sure the python codes are running before you run your MATLAB codes! Besides, please make sure to clear variables just before the functions if need be!*

£ Summing it up, the proposed communication algorithm works perfectly as designed. Please carefully go through the file organizations described in this document for effective implementation of this work. *Never forget to make path-setting changes when trying to run it on a new machine. All paths are set according the organization on my machine. Change COM_PATH on each program (just next the packages and modules sections of p2p1v2.py, p2p2v2.py, & p2p3v2.py) and others on the three configuration files before running the project!*