

PARALLEL & DISTRIBUTED COMPUTING

Assignment # 02

ABSTRACT

This assignment focuses on distributing work load between multiple processes in a cluster and get the work done in less time. MPI Library has been used to implement all tasks.

Muhammad Huzafa

20I-0604

Contents

Cluster Setup	2
Task # 01: Using MPI_Send & MPI_Recv to implement Tasks.....	3
Part # 01: Implementing All Gather using MPI Send & receive	3
Description:.....	3
Output Screenshot	3
Part # 02: Implementing All Gather V using MPI Send and Recive	3
Description:.....	3
Output Screenshot:.....	3
Part # 03: Implementing MPI All to All using MPI Send and Receive	4
Description:.....	4
Output screenshot:	4
Part # 04: Implementing All to All V using MPI Send and Receive	4
Description:.....	4
Output screenshot:	5
Task # 02 : Search & Abort.....	5
Description.....	5
Output Screenshot:.....	5

Cluster Setup

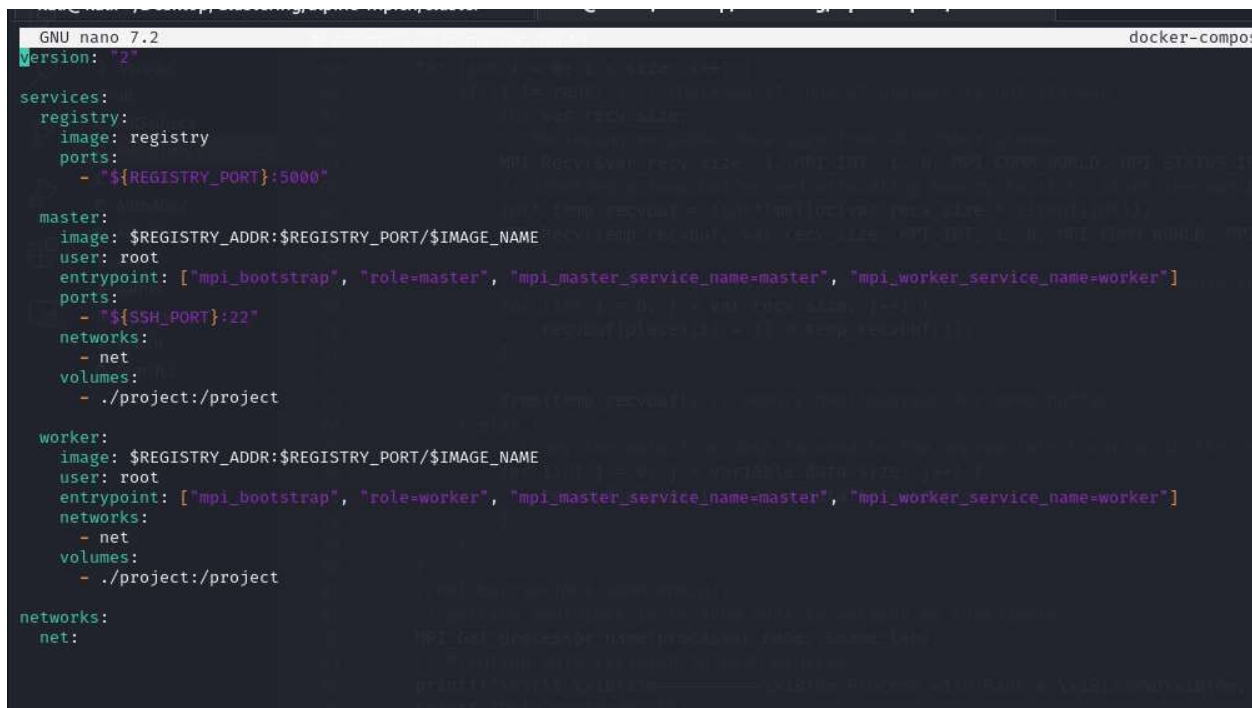
Use the following commands to up your cluster for implementation of the tasks discussed below:

Cd into the cluster folder after downloading and setting up your cluster environment from:

<https://github.com/NLKNGuyen/alpine-mpich>

then traverse into the cluster directory using “cd alpine-mpich/cluster”.

Before beginning the docker for clustering change your docker-compose.yml as shown in the following screenshot to mount your project directory so you do not need to up and down docker every time you update something in your project folder.



```
GNU nano 7.2 docker-compos
version: "2"

services:
  registry:
    image: registry
    ports:
      - "${REGISTRY_PORT}:5000"

  master:
    image: $REGISTRY_ADDR:$REGISTRY_PORT/$IMAGE_NAME
    user: root
    entrypoint: ["mpi_bootstrap", "role=master", "mpi_master_service_name=master", "mpi_worker_service_name=worker"]
    ports:
      - "${SSH_PORT}:22"
    networks:
      - net
    volumes:
      - ./project:/project

  worker:
    image: $REGISTRY_ADDR:$REGISTRY_PORT/$IMAGE_NAME
    user: root
    entrypoint: ["mpi_bootstrap", "role=worker", "mpi_master_service_name=master", "mpi_worker_service_name=worker"]
    networks:
      - net
    volumes:
      - ./project:/project

networks:
  net:
```

Then use the following command to up your cluster:

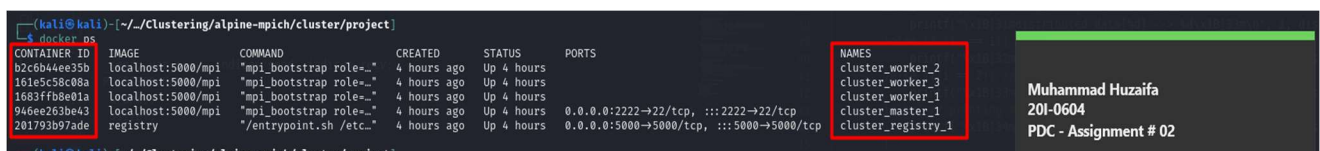
- **Sudo ./cluster.sh up size=<n>** → you can use any size you want
- To down all containers use **sudo ./cluster.sh down size = <n>**
- To run the cluster setup use following command

Sudo ./cluster.sh login

Now if you want to check that if your containers are up or not use the following command:

Docker ps

Using this command, you will get something like this:



```
(kali@kali)~/Clustering/alpine-mpich/cluster/project
$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                               NAMES
b2c6b44ee35b   localhost:5000/mpi                  "mpi_bootstrap role=..." 4 hours ago    Up 4 hours                                     cluster_worker_2
1615c5c808a    localhost:5000/mpi                  "mpi_bootstrap role=..." 4 hours ago    Up 4 hours                                     cluster_worker_3
1683ffb8e01a    localhost:5000/mpi                  "mpi_bootstrap role=..." 4 hours ago    Up 4 hours                                     cluster_master_1
946ee263be43    localhost:5000/mpi                  "mpi_bootstrap role=..." 4 hours ago    Up 4 hours    0.0.0.0:2222->22/tcp, :::2222->22/tcp cluster_registry_1
201793b97ade    registry                            "/entrypoint.sh /etc..." 4 hours ago    Up 4 hours    0.0.0.0:5000->5000/tcp, :::5000->5000/tcp cluster_registry_1

(kali@kali)~/Clustering/alpine-mpich/cluster/project
```

Now that we have our cluster all set up lets get started with our tasks.

Task # 01: Using MPI_Send & MPI_Recv to implement Tasks

Part # 01: Implementing All Gather using MPI Send & receive

Description:

In this code we were supposed to use MPI send and receive to implement all gather MPI function. In short we needed to implement a function where every process in cluster could send its data to every other process and all nodes had all data of all processes.

Output Screenshot

```
/project $ mpirun ./gather
Process with Rank # 0, Container ID # 946ee263be43
Data recieved: [0 1 2 3 ]
Process with Rank # 2, Container ID # b2c6b44ee35b
Process with Rank # 1, Container ID # 1683ffb8e01a
Data recieved: [0 1 2 3 ]
Data recieved: [0]
Process with Rank # 3, Container ID # 161e5c58c08a
Data recieved: [0 1 2 3 ]
1 2 3 ]
/project $
```

Container IDs

Muhammad Huzaifa
20I-0604
PDC - Assignment # 02

As you can see above that all processes have shared their data among all other processes.

Part # 02: Implementing All Gather V using MPI Send and Recive

Description:

This code was all same to the 1st code but instead of sending data of equal size from each process we needed to send data of variable size from each process and at the end all processes should had all data of all processes.

Output Screenshot:

```
/project $ mpicc -o gatherv AllGatherv.c
/project $ mpirun ./gatherv
Printing my data array woth size: 2
20 21
Done printing Data of length 2
Printing my data array woth size: 3
30 31 32
Done printing Data of length 3
Printing my data array woth size: 4
40 41 42 43
Done printing Data of length 4
Printing my data array woth size: 1
Container IDs
Process with Rank # 3, Container ID # 161e5c58c08a
Process with Rank # 2, Container ID # b2c6b44ee35b
Data recieved: [10 20 21 30 31 32 40 41 42 43 ]
Process with Rank # 1, Container ID # 1683ffb8e01a10
Variable Data Recieved by all Processes
Done printing Data of length 10Data recieved: [Data recieved: [
20 20 Data recieved: [21 21 10 30 30 20 31 31 21 32 32 30 40 40 31 41 41 32 42 42 40 43 43 41 ]]42
43 ]
/project $
```

Printing variable size and data for each process

Container IDs

Variable Data Recieved by all Processes

Muhammad Huzaifa
20I-0604
PDC - Assignment # 02

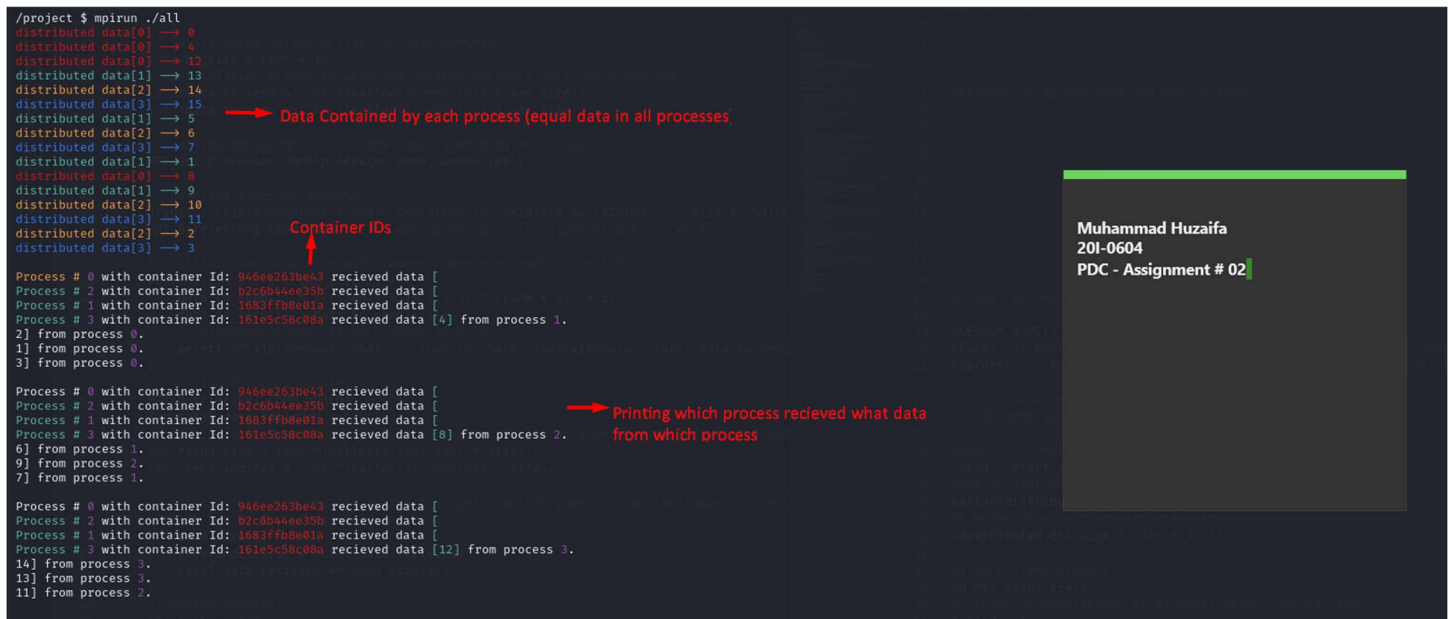
Here in the screenshot you can see that all processes have different sizes and different data but all processes have managed to get the variable size data from all other processes.

Part # 03: Implementing MPI All to All using MPI Send and Receive

Description:

We needed to implement All to All using MPI send and receive. This function basically distributes data in all processes in such a way that each process in the cluster ends up with the portion of data contributed by other processes in the cluster. For instance in array 1 had 1, 2, 3 and array 2 had 4, 5, 6 then proc 1 will have 1 & 4, proc 2 will have 2 & 5 and last proc will have 3 & 6 as shown in the screenshot below:

Output screenshot:



```
/project $ mpirun ./all
distributed data[0] → 0
distributed data[0] → 4
distributed data[0] → 12
distributed data[1] → 13
distributed data[2] → 14
distributed data[3] → 15
distributed data[1] → 5
distributed data[2] → 6
distributed data[3] → 7
distributed data[1] → 1
distributed data[0] → 8
distributed data[1] → 9
distributed data[2] → 10
distributed data[3] → 11
distributed data[2] → 2
distributed data[3] → 3

Process # 0 with container Id: 946ee263be43 recieved data [
Process # 2 with container Id: b2c6b44ee35b recieved data [
Process # 1 with container Id: 1683ffb8e01a recieved data [
Process # 3 with container Id: 161e5c58c08a recieved data [4] from process 1.
2] from process 0.
1] from process 0.
3] from process 0.

Process # 0 with container Id: 946ee263be43 recieved data [
Process # 2 with container Id: b2c6b44ee35b recieved data [
Process # 1 with container Id: 1683ffb8e01a recieved data [
Process # 3 with container Id: 161e5c58c08a recieved data [8] from process 2.
6] from process 1.
9] from process 2.
7] from process 1.

Process # 0 with container Id: 946ee263be43 recieved data [
Process # 2 with container Id: b2c6b44ee35b recieved data [
Process # 1 with container Id: 1683ffb8e01a recieved data [
Process # 3 with container Id: 161e5c58c08a recieved data [12] from process 3.
14] from process 3.
13] from process 3.
11] from process 2.
```

Annotations in the screenshot:

- Red arrow pointing to the first 16 lines of output: "Data Contained by each process (equal data in all processes)"
- Red arrow pointing to the container IDs: "Container IDs"
- Red arrow pointing to the communication lines: "Printing which process recieved what data from which process"

Muhammad Huzaifa
20I-0604
PDC - Assignment # 02

You can see in the screenshot above that each process in cluster have distributed assigned data like 0 has 0, 4, 8, 12 etc. and then its passes this data to other process like shown in lower part of the screenshot where it is shown that which processes has received what data from which process.

Part # 04: Implementing All to All V using MPI Send and Receive

Description:

This is the same as the previous one only diff is that in this function we could send variable data to all other processes using data in distributed manner depending on variable size of data to send and the number of index to send the data from.

Output screenshot:

```
kali@kali: ~/Desktop/Clustering/alpine-mpich/cluster x kali@kali: ~/Desktop/Clustering/alpine-mpich/cluster/project x
/project $ mpicc -o allv AlltoAllv.c
/project $ mpirun ./allv
Process # [3], Container ID:[ 161e5c58c08a ] → Size # [4]
Process # [0], Container ID:[ 946ee263be43 ] → Size # [1]
Process # [1], Container ID:[ 1683ffb8e01a ] → Size # [2]
Process # [2], Container ID:[ b2c6b44ee35b ] → Size # [3]
Rank: [0] → Contains Data: [10]
Rank: [1] → Contains Data: [20]
Rank: [2] → Contains Data: [30]
Rank: [3] → Contains Data: [40]
Rank: [1] → Contains Data: [21]
Rank: [2] → Contains Data: [31]
Rank: [2] → Contains Data: [32]
Rank: [3] → Contains Data: [41]
Rank: [3] → Contains Data: [42]
Rank: [3] → Contains Data: [43]
Process 1 received [42] and [43] from process 3.
Last slave Received: [40] from Process 1
process 3: [40], [41], [21],
process 1: [20], [42], [43],
Master received [30] and [31]
Process 2 received [10] from Master.
process 0: process 2: [30], [10], [31], [31],
```

Verify Container IDs for Cluster

Sizes of all processes

Data contained by each process

Processes after All To All Execution

Data Received from alternative process

Muhammad Huzaifa
20I-0604
PDC - Assignment # 02

As you can see in the screen shot above process1 had 1 element which it sent to process 3 which had 4 elements and process 4 sent 2 elements to process 1 so now both processes have 3 element process 1 →20, 42 & 43 where it received 42 & 43 from process 3 and process 3 has 40, 41 and 20 where it received 20 from process 01. Similar is the case for the 2 other processes.

Task # 02 : Search & Abort

Description

In this task we were suppose to send distributed array to all processes such that if we had an array of 100 we senf 25 element array to 4 process in cluster and then these 4 process will search for an element in respective arrays. If found the element the process sends found signal to the master which on receival sends abort message/ signal to all the processes currently searching in their arrays and they all abort.

Output Screenshot:

```
/project $ mpirun ./search
Target Found - Sending Abort Signal
Process with Rank # 3, Container ID # c62239349849 →67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99,
Aborting Search at process # 3
Process with Rank # 2, Container ID # 9d1ab3b02445 →
Process with Rank # 1, Container ID # 072215a447c3 →34, 1, 35, 2, 36, 3, 37, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
Slave process 1 found the target number.
38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66,
Aborting Search at process # 2
/project $
```

Container IDs

Data in each array

Muhammad Huzaifa
20I-0604
PDC - Assignment # 02

As you can see in the screenshot above that each process finds element in its own array and when found its sends signal to master where master sent signal to all other processes and all other processes aborted search for that element.