# Communication Networks Lab

# Topic Sensor Network-Lab1

1. 附上本次實驗 Q1 4 個 terminals 的結果圖，Q2 溫溼度接收的結果圖

Q1:

```
pi@raspberrypi:~/lab2 $ sudo python Lab2_1_server.py
Worker 1989 is running ...
compute 63 + 9 and send response
compute 57 + 50 and send response
compute 1 + 100 and send response
```

```
pi@raspberrypi:~/lab2 $ sudo python Lab2_1_server.py
Worker 2038 is running ...
compute 91 + 73 and send response
compute 70 + 0 and send response
compute 54 + 44 and send response
```

```
pi@raspberrypi:~/lab2 $ sudo python Lab2_1_server.py
Worker 2087 is running ...
compute 96 + 87 and send response
compute 10 + 72 and send response
compute 95 + 82 and send response
```

```
pi@raspberrypi:~/lab2 $ sudo python Lab2_1_client.py
compute 91 + 73
 = 164 (from worker 2038 )
compute 63 + 9
 = 72 (from worker 1989 )
compute 96 + 87
 = 183 (from worker 2087 )
compute 70 + 0
 = 70 (from worker 2038 )
compute 57 + 50
 = 107 (from worker 1989 )
compute 10 + 72
 = 82 (from worker 2087 )
compute 54 + 44
 = 98 (from worker 2038 )
compute 1 + 100
 = 101 (from worker 1989 )
compute 95 + 82
 = 177 (from worker 2087 )
```

透過 parallel pipeline 的方式將 client 端的 9 筆資料傳給 server 端計算。在 Q1 中共有三個 worker，因此一個 worker 分配到三筆資料，經過計算後將結果回傳給 client。透過分散式的架構，能夠減輕單一 server 的負擔，並且減少計算的時間。

Q2:

```
pi@raspberrypi:~/lab2 $ sudo python Lab2_2.py
Start!!

temp = 74
humidity = 15
temp = -70
humidity = 13
temp = -35
humidity = 27
temp = -58
humidity = 10
temp = -29
humidity = 26
temp = 130
humidity = 36
temp = -37
humidity = 23
temp = 119
humidity = 16
temp = -7
humidity = 24
temp = 134
humidity = 26
avg temperature: 22.1
avg humidity: 21.6
```

助教端為 Publisher，而我們扮演的是 Subscriber。我們訂閱了兩個主題:溫度及濕度，因此我們能夠接收到助教端所傳送的訊息，而後將資料進行後續的處理。(印出來，並且計算平均值)

2. 考慮本次 Q1 和 Q2 的實驗流程下來，請問 ZMQ 的 client 與 server 間的 connect、bind 順序不同的話會對實驗內容有影響嗎?

根據 Q1 的實驗內容，先執行 client.py 而後執行第一個 server.py，再執行第二個 server.py，結果如下:

Client.py

```
pi@raspberrypi:~/lab2 $ sudo python Lab2_1_client.py
compute 43 + 65
 = 108 (from worker 3193 )
compute 5 + 10
 = 15 (from worker 3193 )
compute 35 + 57
 = 92 (from worker 3193 )
compute 68 + 85
 = 153 (from worker 3193 )
compute 5 + 1
 = 6 (from worker 3193 )
compute 19 + 61
 = 80 (from worker 3193 )
compute 30 + 25
 = 55 (from worker 3193 )
compute 35 + 71
 = 106 (from worker 3193 )
compute 88 + 74
 = 162 (from worker 3193 )
```

第一個執行的 server.py

```
pi@raspberrypi:~/lab2 $ sudo python Lab2_1_server.py
Worker 3193 is running ...
compute 43 + 65 and send response
compute 5 + 10 and send response
compute 35 + 57 and send response
compute 68 + 85 and send response
compute 5 + 1 and send response
compute 19 + 61 and send response
compute 30 + 25 and send response
compute 35 + 71 and send response
compute 88 + 74 and send response
```

第二個執行的 server.py

```
pi@raspberrypi:~/lab2 $ sudo python Lab2_1_server.py
Worker 3266 is running ...
```

值得注意的是，因為 client 端只有將 9 筆資料傳送到 server 運算，但是當第一個 server 被開啟時，只有一個 server 能夠進行運算，因此 9 筆資料都是第一個 server 算的。開啟第二個 server 時，他已經沒有東西可以算了。因此可以知道在這樣的情況下，所有資源都是在第一個開啟的 server 進行運算。

但是究竟先開 client 再開 server 真的會影響到分散的架構嗎?因此我又做了一個實驗: 把 client 端發送的運算次數從 9 次增加到 100000 次，去觀察倘若我開啟第一個 server.py 之後還沒把所有運算都完成，那我開啟第二個 server.py 是否還能夠幫忙第一個 server.py 分擔一些運算呢?

更改 client.py 如下: 發送 100000 個運算單位

```python
for i in range(100000):
    a = str(random.randint(0, 100))
    b = str(random.randint(0, 100))

    # Send integer list = [a, b] to servers
    socket.send_multipart([a.encode(),b.encode()])

    # receive results from servers
    result=socket.recv().decode("utf-8")
    print("compute " + a + " + " + b +"\n = " + result)
```

結果(僅截最後幾筆資料):

第一個 server.py(worker 27114)

```
compute 84 + 11 and send response
compute 85 + 78 and send response
compute 73 + 58 and send response
compute 13 + 16 and send response
compute 79 + 75 and send response
compute 72 + 47 and send response
compute 59 + 30 and send response
```

第二個 server.py (worker 27173)

```
compute 96 + 34 and send response
compute 64 + 19 and send response
compute 27 + 48 and send response
compute 20 + 26 and send response
compute 39 + 79 and send response
compute 97 + 97 and send response
compute 15 + 11 and send response
```

Client.py 運算結果:

```
compute 84 + 11
 = 95 (from worker 27114 )
compute 96 + 34
 = 130 (from worker 27173 )
compute 85 + 78
 = 163 (from worker 27114 )
compute 64 + 19
 = 83 (from worker 27173 )
compute 73 + 58
 = 131 (from worker 27114 )
compute 27 + 48
 = 75 (from worker 27173 )
compute 13 + 16
 = 29 (from worker 27114 )
compute 20 + 26
 = 46 (from worker 27173 )
compute 79 + 75
 = 154 (from worker 27114 )
compute 39 + 79
 = 118 (from worker 27173 )
compute 72 + 47
 = 119 (from worker 27114 )
compute 97 + 97
 = 194 (from worker 27173 )
compute 59 + 30
 = 89 (from worker 27114 )
compute 15 + 11
 = 26 (from worker 27173 )
```

可以觀察到當第二個 server.py 被開啟且第一個 server.py 尚未將 client.py 的所有運算單元做完，第二個 server.py 會介入幫忙分散負擔，這是為什麼呢?在一開始 client 便將大量的運算工作分配給第一個 server.py，而為了避免其他還沒連上的 server 沒有工作做，我們在 client.py 加上 time.sleep()以避免其他新連進來的 server 沒有工作做。

```
# wait for all workers connected
time.sleep(1)
```

因此綜合以上的實驗，其實沒有先後順序對結果影響不大，資料都會被完整的回傳給 client.py，只是取決於是誰算的而已。此外，倘若是中途加入的 server，仍然可以幫忙分散運算單元。因此可以推斷，ZeroMQ 對 client 與 server 的啟動順序沒有要求。

## 3. 試想 ZMQ 的 Pub/Sub 可以應用在哪些領域(愈詳細且創新分數越高)

ZMQ 相較其他 message queue 的通信協議有著更高的每秒傳送輸出量，因此我認為可以將一些原本在硬體端(本地端)進行運算的過程傳回公司的主機(server)運算再將其傳回到硬體端。

舉例來說:
特斯拉的車子在不考量網路品質是否穩定的情形，本身這台車就像是一台算力極強的電腦，車況的影像處理，及自駕車的演算法全部都是在本地端執行，因此每台車造價不斐。倘若我們能夠透過 ZMQ 將需要運算的資料都傳回公司的 server 計算完後再回傳到車上顯示給駕駛看的話，那便可以使特斯拉的成本下降許多，並且減少許多功耗。雖然看似可行，但是網路品質會是個最大的問題，倘若網路剛好斷線，而車子正在自動駕駛，那麼車子的行為將會變得不可預測，相當危險。

因此我認為能夠 ZMQ 的應用應該可以換成是在本地端硬體設備較為不好的裝置。例如樹梅派或是其他嵌入式板子加上 sensor，因為這些板子的算力遠不及 server，倘若需要在這些板子上進行一些影像處理甚至是機器學習的演算法勢必不可行，因此我認為能夠透過 ZMQ 將 sensor 或是 camera 讀到的資料傳送到 server 端經過模型或是影像處理後，再將結果傳到需要這些 data 的人，以減少嵌入式板子的運算負擔。

## 4. 本次實驗心得， 你學到了什麼東西?

之前在 topic 3 助教已經先介紹過 socket 及 pub/sub 的基本功能及原理，因此這次的實驗課相對親民，很快就能夠做出來了。不過這次回家在實驗室的工作站進行測試的時候，發現大多 ZMQ 的 code 在透過 socket 傳輸資料時大多還是傳送 byte message，因此都會先 encode，等接收到訊息時再進行 UTF-8 的 decode。感覺這點是 ZMQ 可以加進 API 的一個 function，讓使用者傳任何東西進去，而內部再進行編碼，而傳輸到了目標地之後，再自動進行解碼，將資料傳給接收者。我認為可以加上這個功能讓使用者更方便使用!