

Πανεπιστήμιο Πατρών
Τμήμα Μηχανικών Η/Υ και Πληροφορικής

Κατανεμημένα Συστήματα

Προγραμματιστική Άσκηση για το Σπίτι
Μοντέλα Επικοινωνίας & Συντονισμού για Κατανεμημένα
Συστήματα

Ακαδημαϊκό Έτος 2023/24

Στέφανος-Ιωάννης Πολυδωρόπουλος *

Ζήσης Σούρλας †

3 Μαρτίου 2024

* AM: 1075476 Email: up1075476@ac.upatras.gr

† AM: 1072477 Email: sourlas.zisis@ac.upatras.gr

Περιεχόμενα

1	Μοντέλο Αποστολής-Παράδοσης Μηνυμάτων μέσω Κοινής Ουράς	2
1.1	Παραδείγματα εκτέλεσης	2
1.2	Περιγραφή υλοποίησης	5
1.2.1	Ανάθεση καθήκοντος (task.py)	5
1.2.2	Παραλαβή καθήκοντος (worker.py)	6
1.3	Δυνατότητες και αδυναμίες	6
1.3.1	Ανάθεση καθηκόντων	6
1.3.2	Επιβεβαίωση ανάληψης καθήκοντος	6
1.3.3	Επιβεβαίωση εκτέλεσης καθήκοντος	6
1.3.4	Επιβάρυνση δικτύου	6
1.3.5	Εξισορρόπηση φόρτου εργασίας	7
2	Στοιχειώδες Μοντέλο Ενορχήστρωσης Ανάθεσης Καθηκόντων	7
2.1	Παραδείγματα εκτέλεσης	7
2.2	Περιγραφή υλοποίησης	12
2.2.1	Ανάθεση καθήκοντος (orchestrator.py)	12
2.2.2	Παραλαβή καθήκοντος (worker.py)	12
2.3	Δυνατότητες και αδυναμίες	13
2.3.1	Ανάθεση καθηκόντων	13
2.3.2	Επιβεβαίωση ανάληψης καθήκοντος	13
2.3.3	Επιβεβαίωση εκτέλεσης καθήκοντος	13
2.3.4	Επιβάρυνση δικτύου	13
2.3.5	Εξισορρόπηση φόρτου εργασίας	13
3	Μοντέλο Φιλτραρίσματος Μηνυμάτων	14
3.1	Παραδείγματα εκτέλεσης	14
3.2	Περιγραφή υλοποίησης	17
3.2.1	Ανάθεση καθήκοντος(publisher.py)	18
3.2.2	Παραλαβή καθήκοντος(subscriber.py)	18
3.3	Δυνατότητες και αδυναμίες	18
3.3.1	Ανάθεση καθηκόντων	18
3.3.2	Επιβεβαίωση ανάληψης καθήκοντος	18
3.3.3	Επιβεβαίωση εκτέλεσης καθήκοντος	19
3.3.4	Επιβάρυνση δικτύου	19
3.3.5	Εξισορρόπηση φόρτου εργασίας	19
4	Υπολογισμός Ελάχιστων/Μέγιστων Ενδείξεων Θερμοκρασίας	19
4.1	Παραδείγματα εκτέλεσης	19
4.2	Περιγραφή υλοποίησης	20
4.2.1	Ρουτίνα συντονισμού (HBTG.py)	20
4.2.2	Ρουτίνα κόμβου (node.py)	21

1 Μοντέλο Αποστολής-Παράδοσης Μηνυμάτων μέσω Κοινής Ουράς

1.1 Παραδείγματα εκτέλεσης

Για την εκτέλεση του μοντέλου απαιτείται η εκτέλεση του αρχείου **worker.py** (μία ή περισσότερες φορές) χωρίς ορίσματα ώστε να δημιουργηθούν οι εργάτες. Επιπλέον για κάθε εργασία που θέλουμε να αποσταλεί στην ουρά εργασιών απαιτείται η εκτέλεση του **task.py** επίσης χωρίς ορίσματα.

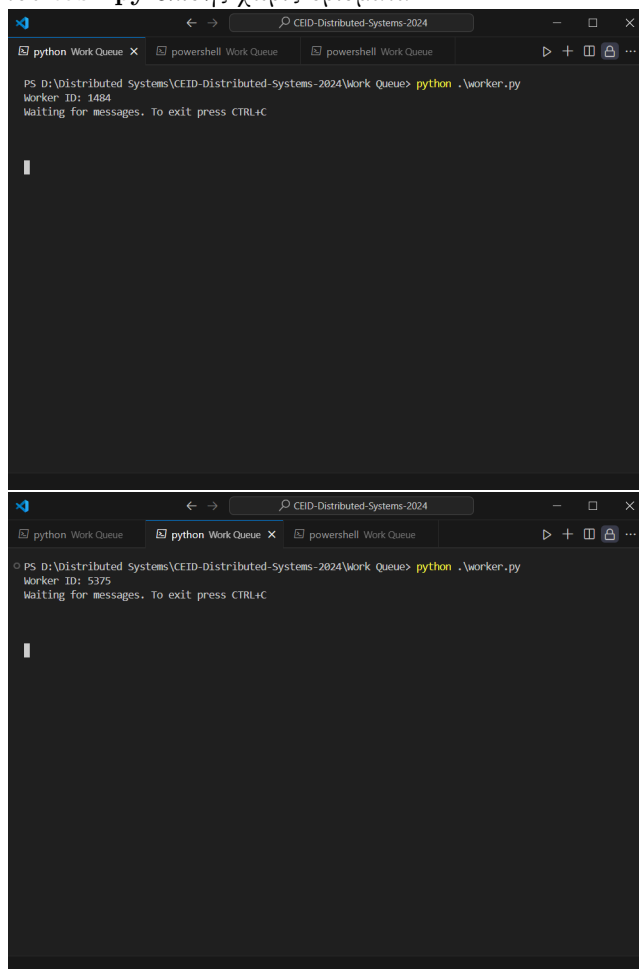


Figure 1: Δημιουργία δύο εργατών

```
CEID-Distributed-Systems-2024
python Work Queue powershell Work Queue X
PS D:\Distributed Systems\CEID-Distributed-Systems-2024\Work Queue> python .\task.py
=====
ID: 6521
Timestamp: 2024-03-01 19:47:37.766093
=====
Message
=====
NEW TASK
PS D:\Distributed Systems\CEID-Distributed-Systems-2024\Work Queue> python .\task.py
=====
ID: 6517
Timestamp: 2024-03-01 19:47:45.754350
=====
Message
=====
NEW TASK
PS D:\Distributed Systems\CEID-Distributed-Systems-2024\Work Queue> python .\task.py
=====
ID: 9834
Timestamp: 2024-03-01 19:47:47.229408
=====
Message
=====
NEW TASK
PS D:\Distributed Systems\CEID-Distributed-Systems-2024\Work Queue> 
```

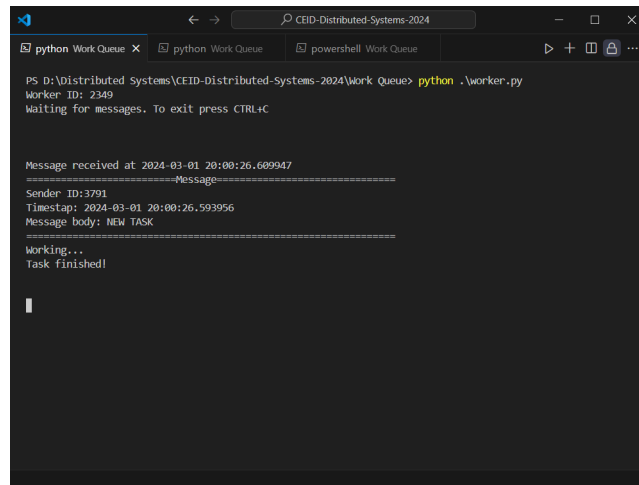
Figure 2: Ανάθεση εργασιών στην ουρά

```
CEID-Distributed-Systems-2024
python Work Queue python Work Queue X powershell Work Queue
PS D:\Distributed Systems\CEID-Distributed-Systems-2024\Work Queue> python .\worker.py
Worker ID: 6816
Waiting for messages. To exit press CTRL+C

Message received at 2024-03-01 20:00:25.757985
=====
Sender ID: 7348
Timestamp: 2024-03-01 20:00:25.742027
Message body: NEW TASK
=====
Working...
Task finished!

Message received at 2024-03-01 20:00:30.760730
=====
Sender ID: 5414
Timestamp: 2024-03-01 20:00:27.395051
Message body: NEW TASK
=====
Working...
Task finished!

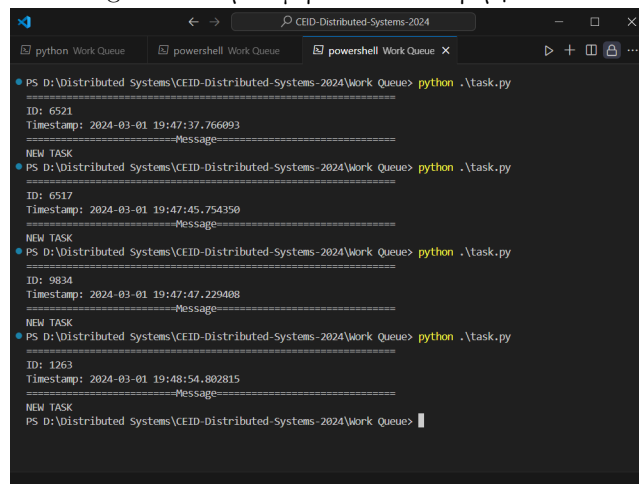

```



```
PS D:\Distributed Systems\CEID-Distributed-Systems-2024\Work Queue> python .\worker.py
Worker ID: 2349
Waiting for messages. To exit press CTRL+C

Message received at 2024-03-01 20:00:26.609947
=====Message=====
Sender ID: 3791
Timestamp: 2024-03-01 20:00:26.593956
Message body: NEW TASK
=====
Working...
Task finished!
```

Figure 3: Παραλαβή και εκτέλεση εργασιών



```
PS D:\Distributed Systems\CEID-Distributed-Systems-2024\Work Queue> python .\task.py
=====Message=====
ID: 6521
Timestamp: 2024-03-01 19:47:37.766093
NEW TASK
PS D:\Distributed Systems\CEID-Distributed-Systems-2024\Work Queue> python .\task.py
=====Message=====
ID: 6517
Timestamp: 2024-03-01 19:47:45.754350
NEW TASK
PS D:\Distributed Systems\CEID-Distributed-Systems-2024\Work Queue> python .\task.py
=====Message=====
ID: 9834
Timestamp: 2024-03-01 19:47:47.229408
NEW TASK
PS D:\Distributed Systems\CEID-Distributed-Systems-2024\Work Queue> python .\task.py
=====Message=====
ID: 1263
Timestamp: 2024-03-01 19:48:54.802815
NEW TASK
PS D:\Distributed Systems\CEID-Distributed-Systems-2024\Work Queue>
```

```
CEID-Distributed-Systems-2024
python Work Queue powershell Work Queue powershell Work Queue
Message received at 2024-03-01 19:48:54.818773
=====Message=====
Sender ID:1263
Timestamp: 2024-03-01 19:48:54.802815
Message body: NEW TASK
=====
Working...
Traceback (most recent call last):
  File "D:\Distributed Systems\CEID-Distributed-Systems-2024\work queue\worker.py", line 46, in <module>
    channel.start_consuming()
  File "D:\Distributed Systems\CEID-Distributed-Systems-2024\venv-win\lib\site-packages\pika\adapters\bloc
king_connection.py", line 1883, in start_consuming
    self._process_data_events(time_limit=None)
  File "D:\Distributed Systems\CEID-Distributed-Systems-2024\venv-win\lib\site-packages\pika\adapters\bloc
king_connection.py", line 2044, in _process_data_events
    self.connection.process_data_events(time_limit=time_limit)
  File "D:\Distributed Systems\CEID-Distributed-Systems-2024\venv-win\lib\site-packages\pika\adapters\bloc
king_connection.py", line 851, in process_data_events
    self._dispatch_channel_events()
  File "D:\Distributed Systems\CEID-Distributed-Systems-2024\venv-win\lib\site-packages\pika\adapters\bloc
king_connection.py", line 567, in _dispatch_channel_events
    impl_channel._get_cookie()._dispatch_events()
  File "D:\Distributed Systems\CEID-Distributed-Systems-2024\venv-win\lib\site-packages\pika\adapters\bloc
king_connection.py", line 1510, in _dispatch_events
    consumer_info.on_message_callback(self, evt.method,
  File "D:\Distributed Systems\CEID-Distributed-Systems-2024\work queue\worker.py", line 38, in callback
    time.sleep(5)
AAAAAAAAAAAAAA

CEID-Distributed-Systems-2024
python Work Queue powershell Work Queue powershell Work Queue
Timestamp: 2024-03-01 19:47:37.766093
Message body: NEW TASK
=====
Working...
Task finished!

Message received at 2024-03-01 19:47:47.245337
=====Message=====
Sender ID:9834
Timestamp: 2024-03-01 19:47:47.229408
Message body: NEW TASK
=====
Working...
Task finished!

Message received at 2024-03-01 19:48:55.983227
=====Message=====
Sender ID:1263
Timestamp: 2024-03-01 19:48:54.802815
Message body: NEW TASK
=====
Working...
Task finished!
```

Figure 4: Παράδειγμα κατάρρευσης εργάτη και παραλαβής εργασίας από άλλον

1.2 Περιγραφή υλοποίησης

Για την υλοποίηση του μοντέλου χρειάστηκε να υλοποιήσουμε δύο προγράμματα. Ένα το οποίο αναθέτει καθήκοντα στην ουρά εργασίας και ένα το οποίο λαμβάνει καθήκοντα από την ουρά εργασίας (εργάτης).

1.2.1 Ανάθεση καθήκοντος (task.py)

Το πρόγραμμα δημιουργεί ένα τυχαίο id αποστολέα, ένα κείμενο μηνύματος και καταγράφει τον χρόνο αποστολής. Αυτά τα αποθηκεύει σε μορφή json και τα κάνει string μέσω της **json.dumps** ώστε να αποσταλούν μέσω του καναλιού. Ανοίγει μια σύνδεση και ένα κανάλι και δημιουργεί μια ουρά με όνομα «tasks_queue». Τέλος αποστέλλει το μήνυμα στην ουρά και κλείνει την σύνδεση.

1.2.2 Παραλαβή καθήκοντος (worker.py)

Το πρόγραμμα δημιουργεί ένα τυχαίο id αποστολέα. Έπειτα ανοίγει μια σύνδεση στο localhost και δημιουργεί μια ουρά με όνομα «tasks_queue». Επιπλέον ορίζει το prefetch_count=1 για το κανάλι ώστε να μη λαμβάνει νέα καθήκοντα όσο εκτελείται ήδη κάποιο. Τέλος αρχίζει να «ακούει» για μηνύματα πάνω στο κανάλι από την συγκεκριμένα ουρά εκτελώντας μια callback συνάρτηση κάθε φορά που λαμβάνει ένα μήνυμα. Αυτή συνάρτηση αποκωδικοποιεί το ληφθέν μήνυμα και ανοίγει το json αρχείο. Αφού εκτυπώσει τα περιεχόμενά του, μπαίνει σε αναμονή για 5 δευτερόλεπτα ώστε να εξομοιώσει την εκτέλεση εργασίας και μετά αποστέλλει μήνυμα ack ώστε να επιβεβαιώσει την εκτέλεση της εργασίας.

1.3 Δυνατότητες και αδυναμίες

1.3.1 Ανάθεση καθηκόντων

Τα καθήκοντα ανατίθενται στους εργάτες μέσω μια ουράς εργασίας. Η αποστολή και η παραλαβή καθηκόντων γίνεται ασύγχρονα. Τα καθήκοντα παραμένουν στην ουρά μέχρι να ανατεθούν σε και να ολοκληρωθούν από κάποιον εργάτη. Εάν δεν υπάρχει κανένας εργάτης διαθέσιμος, τα καθήκοντα παραμένουν στην ουρά μέχρι να υπάρξει. Ακόμα και σε περίπτωση επανεκκίνησης του RabbitMQ Server, τα καθήκοντα δεν πρόκειται να χαθούν καθώς η ουρά έχει οριστεί να είναι durable. Επιπλέον αν ένας εργάτης καταρρεύσει ενώ του έχει ανατεθεί κάποια εργασία ο RabbitMQ Server θα την αναθέσει εκ νέου σε άλλον εργάτη. Συνεπώς, όλα τα καθήκοντα θα ανατεθούν σε εργάτες εν τέλει.

1.3.2 Επιβεβαίωση ανάληψης καθήκοντος

Οι εργάτες δεν αποστέλλουν κάποια επιβεβαίωση κατά την ανάληψη ενός καθήκοντος. Ωστόσο, ο RabbitMQ Server σημειώνει τα καθήκοντα που έχουν αποσταλεί και δεν τα στέλνει σε άλλον εργάτη, παρά μόνο όταν παρέλθει κάποιο χρονικό διάστημα (30 λεπτά από προεπιλογή) χωρίς να λάβει επιβεβαίωση εκτέλεσης ή όταν διαπιστωθεί ότι ο εργάτης έχει καταρρεύσει.

1.3.3 Επιβεβαίωση εκτέλεσης καθήκοντος

Οι εργάτες αποστέλλουν μήνυμα επιβεβαίωσης (ack) αφότου ολοκληρώσουν ένα καθήκον. Μόνο κατόπιν τούτου ο RabbitMQ Server διαγράφει το καθήκον από την ουρά εργασίας.

1.3.4 Επιβάρυνση δικτύου

Το δίκτυο επιβαρύνεται σχεδόν κατά το ελάχιστο για την εξυπηρέτηση των καθηκόντων. Τα μόνα μηνύματα που ανταλλάσσονται μεταξύ ουράς εργασίας και διεργασιών είναι αυτά που αφορούν την αποστολή καθηκόντων από και προς την ουρά και τα μηνύματα επιβεβαίωσης εκτέλεσης καθήκοντος που αποστέλλουν οι εργάτες στην ουρά. Οι μόνες περιπτώσεις μικρότερης επιβάρυνσης του δικτύου

είναι δύο. Πρώτον η παράληψη της ουράς (που οδηγεί σε σύγχρονη επικοινωνία) και δεύτερον η παράληψη της επιβεβαίωσης που δημιουργεί ωστόσο το πρόβλημα της μη εγγυημένης περάτωσης όλων των καθηκόντων.

1.3.5 Εξισορρόπηση φόρτου εργασίας

Ο μόνος τρόπος για να εξισορροπηθεί ως ένα βαθμό ο φόρτος εργασίας μεταξύ των εργατών είναι η μη ανάθεση νέων καθηκόντων σε έναν εργάτη προτού ολοκληρώσει το τρέχον. Αυτό σημαίνει πως αν ένας εργάτης εκτελεί κάποια βαριά εργασία που τον καθυστερεί, δεν θα του ανατεθούν νέες εργασίες αλλά αυτές θα μοιραστούν στους υπόλοιπους εργάτες με round-robin λογική (που αποτελεί προεπιλογή για τον RabbitMQ Server).

2 Στοιχειώδες Μοντέλο Ενορχήστρωσης Ανάθεσης Καθηκόντων

2.1 Παραδείγματα εκτέλεσης

Για την εκτέλεση του συγκεκριμένου μοντέλου επικοινωνίας, απαιτείται η εκτέλεση του αρχείου **worker.py** τόσες φορές όσοι και οι εργάτες που θέλουμε να δημιουργήσουμε για να αναλάβουν την εκτέλεση καθηκόντων. Δίνεται η δυνατότητα στον χρήστη να δώσει μοναδικό, αυστηρά 4-ψήφιο ID για τον κάθε εργάτη. Σε περίπτωση που το ID δεν αποτελείται από 4 ψηφία, εμφανίζεται κατάλληλο μήνυμα λάθους. Διαφορετικά, δημιουργείται τυχαίο ID, όπως και στην προηγούμενη υλοποίηση. Ακόμα, για την δημιουργία καθηκόντων, χρειάζεται να εκτελεστεί το αρχείο **orchestrator.py** (μία ή περισσότερες φορές) όπου δίνονται από τον χρήστη ως παράμετροι οι χρόνοι εκτέλεσης των καθηκόντων (διάρκεια 1-5 δευτερολέπτων). Έτσι, ταυτόχρονα καθορίζεται και ο αριθμός των απαιτούμενων καθηκόντων που θα σταλούν στους εργάτες από την ουρά-ενορχηστρωτή, καθώς και ο χρόνος εκτέλεσης του κάθε καθήκοντος. Δίνεται, όπως και στους εργάτες, η δυνατότητα καθορισμού μοναδικού 4-ψήφιου ID.

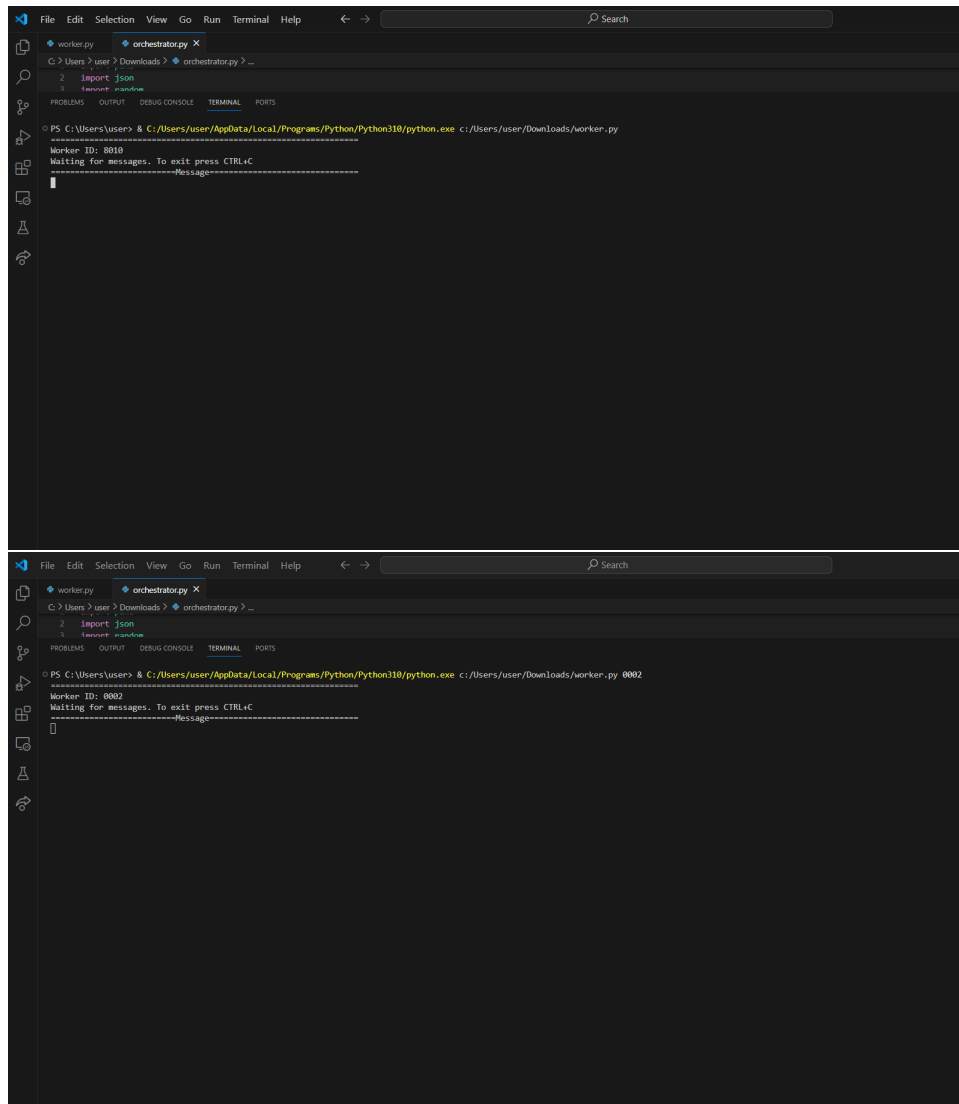


Figure 5: Δημιουργία 2 εργατών

```
File Edit Selection View Go Run Terminal Help
G > Users > user > Downloads > orchestrator.py > ...
2 import json
1 import random

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\user> & C:/Users/user/AppData/Local/Programs/Python/Python310/python.exe c:/Users/user/Downloads/orchestrator.py 0001 1 2 3
=====Message=====
Orchestrator ID: 0001
Sent Orchestrator Task 1 with execution time 1 seconds
Sent Orchestrator Task 2 with execution time 2 seconds
Sent Orchestrator Task 3 with execution time 3 seconds
PS C:\Users\user>

PS C:\Users\user> & C:/Users/user/AppData/Local/Programs/Python/Python310/python.exe c:/Users/user/Downloads/orchestrator.py 0002 3 4 5
=====Message=====
Orchestrator ID: 0002
Sent Orchestrator Task 1 with execution time 3 seconds
Sent Orchestrator Task 2 with execution time 4 seconds
Sent Orchestrator Task 3 with execution time 5 seconds
PS C:\Users\user>
```

Figure 6: Δημιουργία 2 ουρών-ενορχηστρωτών

```
File Edit Selection View Go Run Terminal Help
C:\Users\user> user > Downloads > orchestrator.py > ...
2 import json
3 import random

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Worker ID: 0001
Waiting for messages. To exit press CTRL+C
=====Message=====
Message received at 2024-03-02 16:06:26.974305
=====Message=====
Sender ID:0001
Timestamp: 2024-03-02 16:06:26
Message body: Orchestrator Task 1
=====
Working...
Total Execution Time: █
Task finished in 1 seconds! Total execution time: 1 seconds

Message received at 2024-03-02 16:06:27.990905
=====Message=====
Sender ID:0001
Timestamp: 2024-03-02 16:06:26
Message body: Orchestrator Task 3
=====
Working...
Total Execution Time: █
Task finished in 3 seconds! Total execution time: 4 seconds

Message received at 2024-03-02 16:08:04.696319
=====Message=====
Sender ID:0002
Timestamp: 2024-03-02 16:08:04
Message body: Orchestrator Task 1
=====
Working...
Total Execution Time: ████████
Task finished in 3 seconds! Total execution time: 7 seconds

Message received at 2024-03-02 16:08:07.727080
=====Message=====
Sender ID:0002
Timestamp: 2024-03-02 16:08:04
Message body: Orchestrator Task 3
=====
Working...
Total Execution Time: ██████████
Task finished in 5 seconds! Total execution time: 12 seconds

File Edit Selection View Go Run Terminal Help
C:\Users\user> user > Downloads > orchestrator.py > ...
2 import json
3 import random

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\user> & C:\Users\user\AppData\Local\Programs\Python\Python310\python.exe c:\Users\user\Downloads\worker.py 0002
=====
Worker ID: 0002
Waiting for messages. To exit press CTRL+C
=====Message=====
Message received at 2024-03-02 16:06:26.974305
=====Message=====
Sender ID:0001
Timestamp: 2024-03-02 16:06:26
Message body: Orchestrator Task 2
=====
Working...
Total Execution Time: █
Task finished in 2 seconds! Total execution time: 2 seconds

Message received at 2024-03-02 16:08:04.696319
=====Message=====
Sender ID:0002
Timestamp: 2024-03-02 16:08:04
Message body: Orchestrator Task 2
=====
Working...
Total Execution Time: ████████
Task finished in 4 seconds! Total execution time: 6 seconds
```

Figure 7: Παραλαβή και εκτέλεση εργασιών από τους 2 εργάτες

The image displays two screenshots of a Visual Studio Code (VS Code) terminal window, showing the execution of a Python script and its interaction with a worker.

Top Screenshot:

- The terminal shows the command: `PS C:\Users\User> & C:/Users/user/AppData/Local/Programs/Python/Python310/python.exe c:/Users/user/Downloads/orchestrator.py 2000 4 4 4 5 5 5`
- The output shows the script execution, including the Orchestrator ID (2000) and the execution time of tasks (4 seconds).
- The terminal output is as follows:

```
=====Message=====
Sent Orchestrator Task 1 with execution time 4 seconds
Sent Orchestrator Task 2 with execution time 4 seconds
Sent Orchestrator Task 3 with execution time 4 seconds
Sent Orchestrator Task 4 with execution time 5 seconds
Sent Orchestrator Task 5 with execution time 5 seconds
Sent Orchestrator Task 6 with execution time 5 seconds
=====
PS C:\Users\User>
```

Bottom Screenshot:

- The terminal shows the command: `PS C:\Users\User> & C:/Users/user/AppData/Local/Programs/Python/Python310/python.exe c:/Users/user/Downloads/worker.py 1000`
- The output shows the worker script execution, including the Worker ID (1000), the message received, and the execution time of tasks (4 seconds).
- The terminal output is as follows:

```
Worker ID: 1000
Waiting for messages. To exit press CTRL+C
=====Message=====
Message received at 2024-03-02 16:26:44.023248
=====
Sender ID:2000
Timestamp: 2024-03-02 16:26:44
Message body: Orchestrator Task 1
=====
Working...
Total Execution Time: ████████
Task finished in 4 seconds! Total execution time: 4 seconds

Message received at 2024-03-02 16:26:48.070213
=====Message=====
Sender ID:2000
Timestamp: 2024-03-02 16:26:44
Message body: Orchestrator Task 4
=====
Working...
Total Execution Time: ████████████
Worker interrupted
PS C:\Users\User>
```

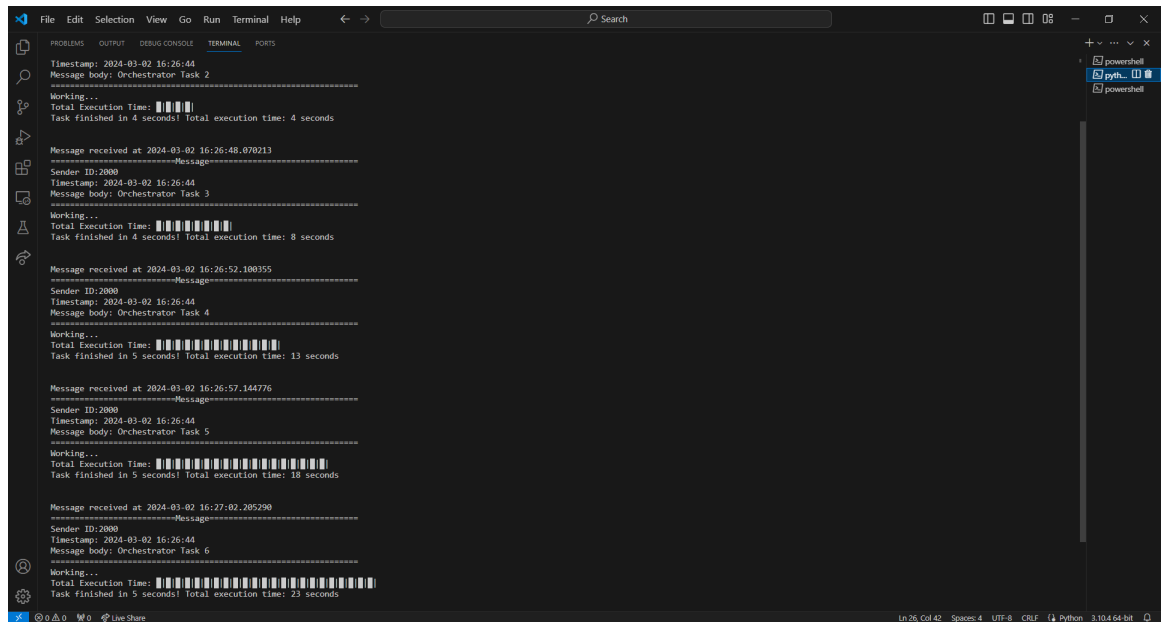


Figure 8: Κατάρρευση εργάτη και παραλαβή καθηκόντων από άλλον

2.2 Περιγραφή υλοποίησης

Για την υλοποίηση του συγκεκριμένου μοντέλου επικοινωνίας, χρειάστηκε η υλοποίηση δύο προγραμμάτων, ένα για την υλοποίηση της ουράς-ενορχηστρωτή, υπεύθυνης για την δημιουργία και τον συντονισμό εργασιών, και την αποστολή τους στους εργάτες για εκτέλεση και ένα για την δημιουργία εργατών, που είναι υπεύθυνοι για τη λήψη εργασιών από την ουρά, την εκτέλεσή τους και την αποστολή επιβεβαίωσης πίσω στον ενορχηστρωτή.

2.2.1 Ανάθεση καθηκόντος (orchestrator.py)

Το πρόγραμμα λαμβάνει το `orchestrator_id` από τη γραμμή εντολών, καθώς και τους χρόνους εκτέλεσης των εργασιών. Γίνονται οι απαραίτητοι έλεγχοι, δηλαδή αν ο `orchestrator_id` είναι έγκυρος 4-ψήφιος αριθμός και αν παρέχεται ένας τουλάχιστον χρόνος εκτέλεσης εργασίας. Δημιουργείται ένας σύνδεσμος με τον RabbitMQ και δηλώνεται μια ουρά με το όνομα `tasks_queue`, όπως και στο προηγούμενο ερώτημα. Στέλνεται κάθε εργασία στην ουρά, παρέχοντας πληροφορίες, όπως το `orchestrator_id`, `timestamp` και χρόνο εκτέλεσης. Εκτυπώνονται πληροφορίες σχετικά με το `orchestrator_id` και κάθε εργασία που αποστέλλει στην ουρά.

2.2.2 Παραλαβή καθηκόντος (worker.py)

Το πρόγραμμα λαμβάνει το `worker_id` από τη γραμμή εντολών ή δημιουργεί ένα τυχαίο, αν δεν παρέχεται από τον χρήστη. Δημιουργείται ένας σύνδεσμος με τον

RabbitMQ και δηλώνεται μια ουρά με το όνομα `tasks_queue`. Η επικοινωνία με τον εντοπιστή γίνεται μέσω JSON, όπως και στην προηγούμενη υλοποίηση. Καθορίζεται κατάλληλη συνάρτηση callback, που καλείται για κάθε εργασία που λαμβάνεται από την ουρά. Οι εργασίες εκτελούνται, εμφανίζονται οι πληροφορίες των εργασιών, και στέλνεται ενημέρωση για τον συνολικό χρόνο εκτέλεσης. Τέλος, στον κώδικα καθορίζεται και κατάλληλο σήμα διακοπής (CTRL+C), με κατάλληλο μήνυμα που ενημερώνει πως τερματίστηκε η λειτουργία του εργάτη.

2.3 Δυνατότητες και αδυναμίες

2.3.1 Ανάθεση καθηκόντων

Σε σχέση με το προηγούμενο μοντέλο, η μόνη διαφορά είναι πως τα καθήκοντα προωθούνται στην ουρά εργασίας μέσω του εντοπιστή, περιλαμβάνοντας και τους φόρτους εργασίας τους. Κατά συνέπεια, ισχύει ο,τι ισχύει και στο προηγούμενο μοντέλο.

2.3.2 Επιβεβαίωση ανάληψης καθήκοντος

Ισχύει ότι και στο προηγούμενο μοντέλο, δηλαδή ότι οι εργάτες δεν αποστέλλουν κάποια επιβεβαίωση κατά την ανάληψη ενός καθήκοντος.

2.3.3 Επιβεβαίωση εκτέλεσης καθήκοντος

Όπως και στο προηγούμενο μοντέλο, οι εργάτες αποστέλλουν μήνυμα επιβεβαίωσης (ack) αφού ολοκληρώσουν ένα καθήκον. Μόνο κατόπιν τούτου ο RabbitMQ Server διαγράφει το καθήκον από την ουρά εργασίας.

2.3.4 Επιβάρυνση δικτύου

Σε σχέση με το προηγούμενο ερώτημα, το δίκτυο, στο συγκεκριμένο μοντέλο επικοινωνίας, επιβαρύνεται ελαφρώς περισσότερο, καθώς τα αντίστοιχα μηνύματα επιβεβαίωσης κάθε εργάτη είναι μεγαλύτερα σε έκταση, αφού δίνουν επιπλέον πληροφορίες σχετικά με την διάρκεια του κάθε καθήκοντος.

2.3.5 Εξισορρόπηση φόρτου εργασίας

Όπως και στην προηγούμενη υλοποίηση, αν ένας εργάτης εκτελεί κάποια βαριά εργασία που τον καθυστερεί, δεν θα του ανατεθούν νέες εργασίες αλλά αυτές θα μοιραστούν στους υπόλοιπους εργάτες με round-robin λογική (που αποτελεί προεπιλογή για τον RabbitMQ Server). Με αυτόν τον τρόπο, διασφαλίζεται η εξισορρόπηση του φόρτου εργασίας.

3 Μοντέλο Φιλτραρίσματος Μηνυμάτων

3.1 Παραδείγματα εκτέλεσης

Για την εκτέλεση του μοντέλου απαιτείται η εκτέλεση των αρχείων **subscriber.py** και **publisher.py** με ορίσματα σύμφωνα με του μορφότυπους που ορίζονται στην εκφώνηση.

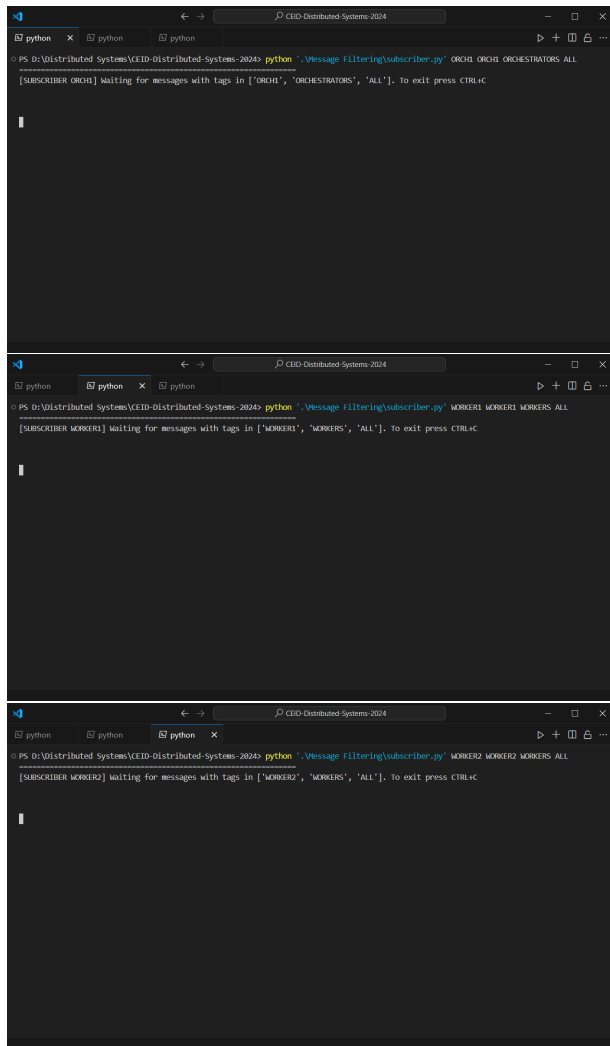


Figure 9: Εγγραφή του ενορχηστρωτή και των εργατών στο ανταλλακτήριο

```
PS D:\Distributed Systems\CEID-Distributed-Systems-2024> python .\Message Filtering\publisher.py ORCH1 WORKERS TASK-ID: ORCH1.1 TASK LOAD 5 seconds
[PUBLISHER ORCH1] Publishing message
-----Message-----
Publisher ID: ORCH1
Message ID: ORCH1.5902
Message Tag: WORKERS
Message body: TASK-ID: ORCH1.1 TASK LOAD 5 seconds
[PUBLISHER ORCH1] New message just published
PS D:\Distributed Systems\CEID-Distributed-Systems-2024>

PS D:\Distributed Systems\CEID-Distributed-Systems-2024> python .\Message Filtering\subscriber.py WORKER1 WORKER1 WORKERS ALL
[SUBSCRIBER WORKER1] waiting for messages with tags in ['WORKER1', 'WORKERS', 'ALL']. To exit press CTRL+C

[SUBSCRIBER WORKER1] Message received at 2024-03-03 22:11:54.276741
-----Message-----
Publisher ID: ORCH1
Message ID: ORCH1.5902
Message Tag: WORKERS
Message body: TASK-ID: ORCH1.1 TASK LOAD 5 seconds
[SUBSCRIBER WORKER1] Total number of messages received: 1
Simulated working for 5 seconds...

PS D:\Distributed Systems\CEID-Distributed-Systems-2024> python .\Message Filtering\subscriber.py WORKER2 WORKER2 WORKERS ALL
[SUBSCRIBER WORKER2] waiting for messages with tags in ['WORKER2', 'WORKERS', 'ALL']. To exit press CTRL+C

[SUBSCRIBER WORKER2] Message received at 2024-03-03 22:11:54.276741
-----Message-----
Publisher ID: ORCH1
Message ID: ORCH1.5902
Message Tag: WORKERS
Message body: TASK-ID: ORCH1.1 TASK LOAD 5 seconds
[SUBSCRIBER WORKER2] Total number of messages received: 1
Simulated working for 5 seconds...
```

Figure 10: Αποστολή αιτήματος διαθεσιμότητας από τον ενορχηστρωτή στους εργάτες

The figure consists of three vertically stacked screenshots of a PowerShell terminal window. The window title is 'CEID-Distributed-Systems-2024'. The first screenshot shows a publisher (WORKER1) publishing a message with ID WORKER1.2290, tag ORCH1, and body 'TASK LOAD: 20 seconds'. The second screenshot shows a publisher (WORKER2) publishing a message with ID WORKER2.4213, tag ORCH1, and body 'TASK LOAD: 20 seconds'. The third screenshot shows a subscriber (ORCH1) waiting for messages with tags in ['ORCH1', 'ORCHESTRATORS', 'ALL']. It receives two messages: one from WORKER1 and one from WORKER2, both with tag ORCH1. The subscriber then simulates working for 5 seconds.

```
PS D:\Distributed Systems\CEID-Distributed-Systems-2024> python ".\Message Filtering\publisher.py" WORKER1 ORCH1 TASK LOAD: 20 seconds  
[PUBLISHER WORKER1] Publishing message  
-----Message-----  
Publisher ID:WORKER1  
Message ID: WORKER1.2290  
Message Tag: ORCH1  
Message body: TASK LOAD: 20 seconds  
-----  
[PUBLISHER WORKER1] New message just published  
PS D:\Distributed Systems\CEID-Distributed-Systems-2024>  
  
PS D:\Distributed Systems\CEID-Distributed-Systems-2024> python ".\Message Filtering\publisher.py" WORKER2 ORCH1 TASK LOAD: 20 seconds  
[PUBLISHER WORKER2] Publishing message  
-----Message-----  
Publisher ID:WORKER2  
Message ID: WORKER2.4213  
Message Tag: ORCH1  
Message body: TASK LOAD: 20 seconds  
-----  
[PUBLISHER WORKER2] New message just published  
PS D:\Distributed Systems\CEID-Distributed-Systems-2024>  
  
python x python python powershell  
[SUBSCRIBER ORCH1] Waiting for messages with tags in ['ORCH1', 'ORCHESTRATORS', 'ALL']. To exit press CTRL-C  
  
[SUBSCRIBER ORCH1] Message received at 2024-03-03 22:18:27.154669  
-----Message-----  
Publisher ID:WORKER1  
Message ID: WORKER1.2290  
Message Tag: ORCH1  
Message body: TASK LOAD: 20 seconds  
-----  
[SUBSCRIBER ORCH1] Total number of messages received: 1  
Simulated working for 5 seconds...  
[SUBSCRIBER ORCH1] Message received at 2024-03-03 22:18:58.868467  
-----Message-----  
Publisher ID:WORKER2  
Message ID: WORKER2.4213  
Message Tag: ORCH1  
Message body: TASK LOAD: 20 seconds  
-----  
[SUBSCRIBER ORCH1] Total number of messages received: 2  
Simulated working for 5 seconds...
```

Figure 11: Επιβεβαίωση διαθεσιμότητας από τους εργάτες

The screenshot shows a PowerShell terminal window with the title 'CEID-Distributed-Systems-2024'. The terminal displays the output of a publisher (ORCH1) publishing a message with ID ORCH1.1039, tag WORKER1, and body 'ASSIGNED TASK ID: ORCH1.1 TASK LOAD 5 seconds'. The subscriber (ORCH1) is waiting for messages with tags in ['ORCH1', 'ORCHESTRATORS', 'ALL']. It receives the message and then simulates working for 5 seconds.

```
PS D:\Distributed Systems\CEID-Distributed-Systems-2024> python ".\Message Filtering\publisher.py" ORCH1 WORKER1 ASSIGNED TASK ID: ORCH1.1 TASK LOAD 5 seconds  
[PUBLISHER ORCH1] Publishing message  
-----Message-----  
Publisher ID:ORCH1  
Message ID: ORCH1.1039  
Message Tag: WORKER1  
Message body: ASSIGNED TASK ID: ORCH1.1 TASK LOAD 5 seconds  
-----  
[PUBLISHER ORCH1] New message just published  
PS D:\Distributed Systems\CEID-Distributed-Systems-2024>
```

```
[SUBSCRIBER WORKER1] waiting for messages with tags in ['WORKER1', 'WORKERS', 'ALL']. To exit press CTRL+C

[SUBSCRIBER WORKER1] Message received at 2024-03-03 22:11:54.276741
-----Message-----
Publisher ID: ORCH1
Message ID: ORCH1.5902
Message Tag: WORKERS
Message body: TASK-ID: ORCH1.1 TASK LOAD 5 seconds
[SUBSCRIBER WORKER1] Total number of messages received: 1
Simulated working for 5 seconds...

[SUBSCRIBER WORKER1] Message received at 2024-03-03 22:20:38.119534
-----Message-----
Publisher ID: ORCH1
Message ID: ORCH1.1039
Message Tag: WORKER1
Message body: ASSIGNED TASK-ID: ORCH1.1 TASK LOAD 5 seconds
[SUBSCRIBER WORKER1] Total number of messages received: 2
Simulated working for 5 seconds...
```

Figure 12: Ανάθεση καθήκοντος σε εργάτη

```
PS D:\Distributed Systems\CEID-Distributed-Systems-2024> python .\Message Filtering\publisher.py WORKER1 ORCH1 COMPLETED TASK-ID: ORCH1.1
[PUBLISHER WORKER1] Publishing message
-----Message-----
Publisher ID: WORKER1
Message ID: WORKER1.3753
Message Tag: ORCH1
Message body: COMPLETED TASK-ID: ORCH1.1
[PUBLISHER WORKER1] New message just published
PS D:\Distributed Systems\CEID-Distributed-Systems-2024>

[SUBSCRIBER ORCH1] Total number of messages received: 1
Simulated working for 5 seconds...

[SUBSCRIBER ORCH1] Message received at 2024-03-03 22:18:58.868467
-----Message-----
Publisher ID: WORKER2
Message ID: WORKER2.4213
Message Tag: ORCH1
Message body: TASK LOAD: 20 seconds
[SUBSCRIBER ORCH1] Total number of messages received: 2
Simulated working for 5 seconds...

[SUBSCRIBER ORCH1] Message received at 2024-03-03 22:22:43.428054
-----Message-----
Publisher ID: WORKER1
Message ID: WORKER1.3753
Message Tag: ORCH1
Message body: COMPLETED TASK-ID: ORCH1.1
[SUBSCRIBER ORCH1] Total number of messages received: 3
Simulated working for 5 seconds...
```

Figure 13: Αποστολή επιβεβαίωση εκτέλεσης καθήκοντος από τον εργάτη στον εντορηστροτή

3.2 Περιγραφή υλοποίησης

Για την υλοποίηση του μοντέλου επικοινωνίας μεταξύ πολλαπλών δημοσιευτών και συνδρομητών, χρειάστηκε να υλοποιήσουμε 2 προγράμματα, ένα για την υλοποίηση του publisher, του υπεύθυνου δηλαδή για την δημοσίευση των μηνυμάτων σε ένα ανταλλακτήριο προς τους συνδρομητές και του subscriber, υπεύθυνου για την εκτέλεση εργασιών με βάση συγκεκριμένες ετικέτες (TAGS).

3.2.1 Ανάθεση καθήκοντος(publisher.py)

Το πρόγραμμα δέχεται ως παραμέτρους το αναγνωριστικό του publisher το tag προς το οποίο αποστέλλει και το περιεχόμενο του μηνύματος. Ένα μοναδικό αναγνωριστικό μηνύματος **msg_ID** δημιουργείται, συνδυάζοντας το μοναδικό αναγνωριστικό του publisher και ένα τυχαίο ακέραιο. Δημιουργείται σύνδεση και δηλώνεται ένα θεματικό ανταλλακτήριο, το **task_stream**. Δημοσιεύεται το μήνυμα στο ανταλλακτήριο με το routing key το tag που δόθηκε ως είσοδος. Το μήνυμα περιλαμβάνει το μοναδικό αναγνωριστικό του publisher, τις ετικέτες, καθώς και το σώμα του μηνύματος. Τέλος, το πρόγραμμα εκτυπώνει πληροφορίες σχετικά με το δημοσιευμένο μήνυμα.

3.2.2 Παραλαβή καθήκοντος(subscriber.py)

Το πρόγραμμα δέχεται ως παραμέτρους το αναγνωριστικό του subscriber και άλλες παραμέτρους που παρέχουν πληροφορίες χρήσιμες για τους συνδρομητές. Γίνεται σύνδεση και δηλώνεται το ίδιο θεματικό ανταλλακτήριο, το **task_stream**. Κάθε συνδρομητής συνδέεται με το ανταλλακτήριο με τις καθορισμένες ετικέτες. Με αυτόν τον τρόπο, εξασφαλίζεται πως κάθε συνδρομητής θα λάβει μόνο μηνύματα που τον ενδιαφέρουν. Ορίζεται μια **callback** συνάρτηση, η οποία εκτελείται κάθε φορά που λαμβάνεται ένα μήνυμα. Η συνάρτηση αποκωδικοποιεί το μήνυμα, εκτυπώνει τα περιεχόμενά του, προσομοιάζει κάποιο χρόνο επεξεργασίας, αυξάνει τον μετρητή μηνυμάτων και εκτυπώνει την τιμή του.

3.3 Δυνατότητες και αδυναμίες

3.3.1 Ανάθεση καθήκοντων

Στην συγκεκριμένη υλοποίηση, ο ενορχηστρωτής αποστέλλει μήνυμα προς όλους τους εργάτες, ως αίτημα διαθεσιμότητας. Στο αίτημα αυτό, απαντούν όσοι εργάτες είναι ενεργοί, δηλαδή εγγεγραμμένοι στο ανταλλακτήριο FILTERING STREAM, αποστέλλοντας το TOTAL LOAD τους σαν επιβεβαίωση ότι είναι διαθέσιμοι. Έτσι, ο ενορχηστρωτής θα αναθέσει τα κατάλληλα καθήκοντα στον εκάστοτε εργάτη, ανάλογα με τα ID τους. Όμως, αν κατά την διάρκεια ανάθεσης μιας εργασίας, ο εργάτης καταρρεύσει, ο ενορχηστρωτής δεν μπορεί να αναθέσει την ίδια εργασία σε κάποιον άλλον εργάτη, αφού δεν δουλεύει ως ουρά. Στην συγκεκριμένη περίπτωση, η εργασία αυτή θα χαθεί. Ακόμα, αν κανένας εργάτης δεν είναι ενεργός, δεν επιβεβαιώνεται ποτέ το αίτημα διαθεσιμότητας του ενορχηστρωτή και επομένως δεν θα εκτελεστεί κάποια εργασία αφού το μήνυμα θα χαθεί καθώς δεν διατηρείται σε ουρά. Αυτές οι 2 περιπτώσεις δείχνουν πως δεν είναι βέβαιο πως θα ανατεθούν όλα τα καθήκοντα σε εργάτες.

3.3.2 Επιβεβαίωση ανάληψης καθήκοντος

Όπως και στις προηγούμενες υλοποιήσεις, οι εργάτες δεν αποστέλλουν κάποια επιβεβαίωση, όταν αναλαμβάνουν κάποια εργασία. Ωστόσο, αποστέλλουν αίτημα διαθεσιμότητας όταν τους ανακοινώνεται κάποια εργασία.

3.3.3 Επιβεβαίωση εκτέλεσης καθήκοντος

Μόλις κάποιος ενεργός εργάτης ολοκληρώσει την εκτέλεση κάποιας εργασίας, αποστέλλει μήνυμα επιβεβαίωσης εκτέλεσης στον ενορχηστρωτή (COMPLETED TASK ID). Ωστόσο, ελλείψει αυτού του μηνύματος, ο ενορχηστρωτής δεν θα αναθέσει σε άλλον εργάτη την ίδια εργασία σε αντίθεση με ό,τι συμβαίνει στην ουρά εργασίας.

3.3.4 Επιβάρυνση δικτύου

Το δίκτυο, στην συγκεκριμένη υλοποίηση, επιβαρύνεται σαφώς περισσότερο, καθώς αποστέλλονται περισσότερα μηνύματα, τα οποία αφορούν τα αιτήματα διαθεσιμότητας εργατών από τον ενορχηστρωτή, τα αντίστοιχα μηνύματα επιβεβαίωσης από τους εργάτες, καθώς και τα μηνύματα ανάληψης καθήκοντων από τους ενορχηστρωτές που στέλνονται ξεχωριστά σε κάθε ενεργό εργάτη και τα μηνύματα επιβεβαίωσης εκτέλεσης εργασιών από τους εργάτες.

3.3.5 Εξισορρόπηση φόρτου εργασίας

Είναι εφικτή η εξισορρόπηση του φόρτου εργασίας μεταξύ των ενεργών εργατών λόγω της πολιτικής ανάθεσης των καθήκοντων που δίνεται από την εκφώνηση. Συγκεκριμένα, ο ενορχηστρωτής, εφόσον οργανώνει τα καθήκοντα σε φθίνουσα σειρά χρόνων εκτέλεσης και τα αναθέτει στους ενεργούς εργάτες με τέτοιο τρόπο, ώστε να επιτυγχάνεται η ελαχιστοποίηση του μέγιστου συνολικού φόρτου τους, συμπεραίνεται πως κατανέμονται με δίκαιο τρόπο τα καθήκοντα και κανένας εργάτης δεν θα εργάζεται πολύ παραπάνω ή κάποιος πολύ λιγότερο.

4 Υπολογισμός Ελάχιστων/Μέγιστων Ενδείξεων Θερμοκρασίας

4.1 Παραδείγματα εκτέλεσης

Η εκτέλεση του δικτύου γίνεται να γίνει με δύο τρόπους: αυτόματα ή χειροκίνητα. Εκτελώντας το script **run.ps1** εκτελείται αυτόματα το **node.py** 9 φορές (για όλους τους κόμβους) με τα κατάλληλα ορίσματα. Στη συνέχεια εκτελείται το **HBTG.py** με όρισμα τη μη ανανέωση των μετρήσεων σε κάθε κύκλο. Για να εκτελεστεί χειροκίνητα το δίκτυο απαιτείται η εκτέλεση του **node.py** διαδοχικά δίνοντας ως όρισμα κάθε φορά το αναγνωριστικό του κόμβου. Επιπλέον απαιτείται η εκτέλεση του **HBTG.py** μία φορά με όρισμα 1 ή 0 ανάλογα με το αν επιθυμούμε να ανανεώνονται οι μετρήσεις ή όχι.

```
Windows PowerShell
Min Global Temp: 0.0
Avg Global Temp: 10.719223616302394
Max Global Temp: 38.5

[NODE 14] Temps received
=====
HB[5]
=====
Min Local Temp: 0.0
Avg Local Temp: 15.575
Max Local Temp: 29.5

Min Global Temp: 0.0
Avg Global Temp: 10.772914182256038
Max Global Temp: 38.5

[NODE 14] Temps received
=====
HB[6]
=====
Min Local Temp: 0.0
Avg Local Temp: 25.575
Max Local Temp: 29.8

Min Global Temp: 0.0
Avg Global Temp: 10.7931163333042
Max Global Temp: 38.5

Windows PowerShell
Min Global Temp: 0.0
Avg Global Temp: 10.758064962533819
Max Global Temp: 38.5

[NODE 28] Temps received
=====
HB[5]
=====
Min Local Temp: 5.2
Avg Local Temp: 16.78
Max Local Temp: 37.4

Min Global Temp: 0.0
Avg Global Temp: 11.63460004265147
Max Global Temp: 38.5

[NODE 28] Temps received
=====
HB[6]
=====
Min Local Temp: 5.2
Avg Local Temp: 26.78
Max Local Temp: 37.4

Min Global Temp: 0.0
Avg Global Temp: 10.852720004346725
Max Global Temp: 38.5

Windows PowerShell
Min Global Temp: 0.0
Avg Global Temp: 10.100285942924964
Max Global Temp: 38.5

[NODE 18] Temps received
=====
HB[5]
=====
Min Local Temp: 11.4
Avg Local Temp: 11.7
Max Local Temp: 12.0

Min Global Temp: 0.0
Avg Global Temp: 10.4020572876755
Max Global Temp: 38.5

[NODE 18] Temps received
=====
HB[6]
=====
Min Local Temp: 11.4
Avg Local Temp: 11.7
Max Local Temp: 12.0

Min Global Temp: 0.0
Avg Global Temp: 10.152867939495776
Max Global Temp: 38.5

Windows PowerShell
HB[6]
=====
Temps sent to [NODE 11]
=====
HB[6]
=====
Temps sent to [NODE 14]
=====
HB[6]
=====
Temps sent to [NODE 18]
=====
HB[6]
=====
Temps sent to [NODE 20]
=====
HB[6]
=====
Temps sent to [NODE 21]
=====
HB[6]
=====
Temps sent to [NODE 28]
```

Figure 14: Παράδειγμα εκτέλεσης του δικτύου

4.2 Περιγραφή υλοποίησης

Για την δημιουργία του δικτύου επικάλυψης και την παραγωγή μετρήσεων θερμοκρασίας υλοποιήσαμε ένα μοντέλο το οποίο αποτελείται από 2 ανταλλακτήρια και μία ουρά εργασίας. Το ένα ανταλλακτήριο (**temp_stream**) χρησιμοποιείται για την αποστολή των θερμοκρασιών από την ρουτίνα παραγωγής παλμών στους κόμβους, ενώ το δεύτερο (**node_stream**) χρησιμοποιείται από τους κόμβους ώστε να ανταλλάσσουν τις καθολικές μετρήσεις θερμοκρασιών μεταξύ τους. Τέλος η ουρά εργασίας χρησιμοποιείται ώστε να στέλνονται μηνύματα επιβεβαίωσης για την λήψη των θερμοκρασιών από τους κόμβους στη ρουτίνα παραγωγής παλμών.

Για την υλοποίηση αυτών χρειάστηκαν δύο προγράμματα. Ένα που υλοποιεί την ρουτίνα συντονισμού (**HBTG.py**) και ένα που υλοποιεί τον κόμβο (**node.py**).

4.2.1 Ρουτίνα συντονισμού (HBTG.py)

Το πρόγραμμα διατηρεί μια λίστα με τους κόμβους του δικτύου και μια λίστα με τους αισθητήρες του κάθε κόμβου. Επίσης διατηρεί τον αριθμό του τρέχοντος παλμού και έναν μετρητή απαντήσεων από τους κόμβους. Οι λειτουργίες του υλοποιούνται από μια σειρά συναρτήσεων.

main() Η συγκεκριμένη συνάρτηση παράγει για κάθε κόμβο (με τυχαίο τρόπο) τόσες τιμές θερμοκρασιών όσες και οι αισθητήρες του. Στη συνέχεια τις αποθηκεύει σε μορφή json μαζί με τον αριθμό του τρέχοντος παλμού και τις αποστέλλει στους κόμβους μέσω της **temps_publish**. Έπειτα μπαίνει σε sleep mode

για 10 δευτερόλεπτα και στη συνέχεια ελέγχει αν έλαβε όσες απαντήσεις όσες και οι κόμβοι του δικτύου. Όταν διαπιστωθεί ότι απάντησαν όλοι οι κόμβοι, αυξάνει τον αριθμό του παλμού, μηδενίζει τον μετρητή απαντήσεων και επαναλαμβάνει την πιο πάνω διαδικασία δημιουργώντας νέες μετρήσεις ή διατηρώντας τις ίδιες ανάλογα με το όρισμα που έλαβε στην είσοδο.

temps_publish(id, message) Η συνάρτηση λαμβάνει ως είσοδο το αναγνωριστικό ενός κόμβου και το μήνυμα που πρέπει να αποστείλει. Δημιουργεί μια σύνδεση με το ανταλλακτήριο temp_stream και αποστέλλει το μήνυμα με routing_key το αναγνωριστικό του κόμβου. Παράλληλα τυπώνει στο τερματικό κατάλληλο μήνυμα.

acks_subscribe Η συνάρτηση δημιουργεί μια σύνδεση με την ουρά εργασίας acks_queue από την οποία λαμβάνει μηνύματα επιβεβαίωσης από τους κόμβους. Κάθε φορά που λαμβάνει ένα μήνυμα καλεί την **acks_callback**

acks_callback Η συνάρτηση αποκωδικοποιεί το μήνυμα που ελήφθη και ελέγχει αν αντιστοιχεί στον τρέχοντα παλμό. Αν ισχύει αυτό τότε αυξάνει τον μετρητή απαντήσεων κατά 1.

4.2.2 Ρουτίνα κόμβου (node.py)

Η ρουτίνα επιλέγει τυχαία έναν αριθμό που αποτελεί το πλήθος των έξω γειτόνων του κόμβου. Επιλέγει επίσης τυχαία αντίστοιχο αριθμό κόμβων ώστε να αποτελέσουν τους έξω γείτονες. Οι λειτουργίες του υλοποιούνται από μια σειρά συναρτήσεων.

temps_subscribe Η συνάρτηση ανοίγει μια σύνδεση με το ανταλλακτήριο temp_stream από το οποίο λαμβάνει μηνύματα με routing key το αναγνωριστικό του κόμβου. Κάθε φορά που λαμβάνει ένα μήνυμα καλεί την **temps_callback**.

temps_callback Η συνάρτηση αποκωδικοποιεί το μήνυμα που ελήφθη και ενημερώνει βάσει των περιεχομένων την τοπική τριάδα ελάχιστης/μέσης/μέγιστης θερμοκρασίας. Στη συνέχεια ενημερώνει και τις καθολικές θερμοκρασίες βάσει των νέων τοπικών. Αν έχει επέλθει αλλαγή στις καθολικές θερμοκρασίες, αποστέλλει στους έξω γείτονες του κόμβου τις νέες καθολικές θερμοκρασίες που υπολόγισε καλώντας την **node_publish**. Στην συνέχεια τυπώνει στο τερματικό τις τοπικές και καθολικές θερμοκρασίες. Τέλος συνδέεται στην ουρά acks_queue και αποστέλλει το αναγνωριστικό του μαζί με τον παλμό στον οποίο αναφέρεται εν είδει μηνύματος επιβεβαίωσης.

node_publish Η συνάρτηση δέχεται ως όρισμα το μήνυμα που πρέπει να αποστείλει. Δημιουργεί μια σύνδεση με το ανταλλακτήριο node_stream και αποστέλλει το μήνυμα σε όλους τους έξω γείτονες του κόμβου, χρησιμοποιώντας ως routing_key τα αναγνωριστικά τους.

node_subscribe Η συνάρτηση δημιουργεί μια σύνδεση με το ανταλλακτήριο `node_stream` και εγγράφεται για λήψη μηνυμάτων που έχουν ως routing key το αναγνωριστικό του κόμβου. Όταν ληφθεί κάποιο μήνυμα καλείται η **node_callback**. Μέσω αυτής της συνάρτησης ο κόμβος λαμβάνει, από τους κόμβους στους οποίους είναι έξω-γείτονας, τις καθολικές τιμές τους.

node_callback Η συνάρτηση αποκωδικοποιεί το μήνυμα που ελήφθη και ενημερώνει βάσει των περιεχομένων την καθολική τριάδα ελάχιστης/μέσης/μέγιστης θερμοκρασίας. Αν έχει επέλθει αλλαγή στις καθολικές θερμοκρασίες, αποστέλλει στους έξω γείτονες του κόμβου τις νέες καθολικές θερμοκρασίες που υπολόγισε καλώντας την **node_publish**.