Υπολογιστικά Νέφη
ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ 4
« Σύγκριση απόδοσης συστήματος τεσσάρων παράλληλων επεξεργαστών »
Οκτώβριος 2024

Συμπληρώστε τα στοιχεία σας πριν την αποστολή της εργασίας	
Ονοματεώπυνο	Δημήτριος Γκούμας
Αριθμός Μητρώου	4502

Κατανεμημένη – Παράλληλη επεξεργασία

Ένα κατανεμημένο σύστημα αποτελείται από μια συλλογή ανεξάρτητων συστημάτων (κόμβων) τα οποία μοιράζονται κάποιους πόρους. Τα κατανεμημένα συστήματα γίνονται όλο και πιο συχνά επειδή παρέχουν την δυνατότητα για αυξημένη υπολογιστική ισχύ, έχουν μεγάλη αποτελεσματικότητα κόστους και μπορούν να επεκταθούν εύκολα.

Τα συστήματα αυτά είναι απαραίτητα για να ικανοποιηθούν οι ανάγκες των απαιτητικών εφαρμογών των διαφόρων επιστημονικών πεδίων. Ακόμα και τα πιο ισχυρά υπολογιστικά περιβάλλοντα υψηλής απόδοσης **χρειάζονται κάποιο κατάλληλο χρονοπρογραμματισμό (scheduling)**, για να επιτύχουν την επιθυμητή ποιότητα υπηρεσίας τόσο από την μεριά του συστήματος όσο και του χρήστη.

Για τον λόγο αυτό πολλοί ερευνητές αναζητούν μεθόδους χρονοπρογραμματισμού οι οποίες θα βελτιστοποιούν την απόδοση του κατανεμημένου συστήματος. Επιπλέον, οι εξελίξεις στα υπολογιστικά περιβάλλοντα, όπως τα grid και cloud περιβάλλοντα και τα ετερογενή clusters, παρουσιάζουν νέες προκλήσεις χρονοπρογραμματισμού.

Η ετερογένεια αυτών των συστημάτων και η μεταβλητότητα των πόρων (επεξεργαστές ή ομάδες επεξεργαστών μπορούν να προστεθούν ή να αφαιρεθούν οποιαδήποτε στιγμή) απαιτεί συχνά αφοσιωμένες τεχνικές χρονοπρογραμματισμού για κάθε ομάδα πόρων του κατανεμημένου υπολογιστικού περιβάλλοντος.

Ο χρονοπρογραμματισμός σε κατανεμημένα συστήματα έχει μελετηθεί εκτενώς προσπαθώντας να επιτύχει διαφόρους στόχους απόδοσης σε πολλούς τύπους κατανεμημένων συστημάτων και παράλληλου φόρτου εργασίας. Είναι γενικά παραδεκτό ότι ο χρονοπρογραμματισμός είναι θεμελιώδης για την επιτυχία των κατανεμημένων συστημάτων.

Σήμερα υπάρχουν διάφοροι τύποι κατανεμημένων συστημάτων, τα περισσότερα από τα οποία αποτελούνται από ένα μεγάλο αριθμό επεξεργαστών. Η πληθώρα κατανεμημένων συστημάτων όπως επίσης και η μεγάλη ποικιλία **σε υλικό (hardware) και λογισμικό**

(software) κάνουν την διαχείριση των πόρων ενός κατανεμημένου συστήματος εξαιρετικά δύσκολη.

Ο χρονοπρογραμματισμός των παράλληλων tasks μιας εργασίας (job) είναι τις περισσότερες φορές ένα σύνθετο πρόβλημα που καθορίζει τόσο για το πότε θα εκτελεστεί μια διεργασία αλλά και πού θα εκτελεστεί. Σύμφωνα με αυτό, ο χρονοπρογραμματισμός στα κατανεμημένα συστήματα αποτελείται από δύο στοιχεία:

- τον ***διανεμητή*** (allocator): που αποφασίζει για το **πού** θα εκτελεστεί ένα task
- τον ***χρονοπρογραμματιστή*** (scheduler), που αποφασίζει **πότε** θα πάρει ένα task το μερίδιο του επεξεργαστή που του ανήκει.

Τυπικά κάθε κόμβος σε ένα κατανεμημένο σύστημα:

- **έχει τον δικό του χρονοπρογραμματιστή**, για να χρονοπρογραμματίζει τα tasks στον τοπικό επεξεργαστή, συνήθως με τρόπους διαμοιρασμού του χρόνου, ενώ
- **ο διανεμητής λαμβάνει αποφάσεις υψηλότερου επιπέδου** για την εκχώρηση των διεργασιών και αποτελεί συνήθως ένα ξεχωριστό κόμβο.

Παρόλο που υπάρχουν αρκετές παραλλαγές, αυτό το σχήμα δείχνει να είναι πιο φυσικό για τα κατανεμημένα συστήματα. Ο λόγος για αυτό είναι διπλός.

- **Πρώτον, κάθε κόμβος έχει συνήθως το δικό του λειτουργικό** σύστημα το οποίο είναι ικανό για χρονοπρογραμματισμό των διεργασιών.
- **Ο δεύτερος λόγος είναι ο ευκολότερος σχεδιασμός**, αφού οι σχεδιαστές ενός κατανεμημένου συστήματος μπορούν να εστιάσουν καλύτερα στο πολύπλοκο πρόβλημα της διανομής του φόρτου χωρίς να επιβαρύνονται με την κάθε λεπτομέρεια του χρονοπρογραμματισμού.

Ο διανεμητής και ο χρονοπρογραμματιστής **εφαρμόζουν τις πολιτικές ανάθεσης** των tasks και χρονοπρογραμματισμού αντίστοιχα. Η πολιτική ανάθεσης προσπαθεί **να κατανείμει τον συνολικό φόρτο** του συστήματος στους ανεξάρτητους κόμβους μεταφέροντας τις εργασίες στους κόμβους.

- **Η πολιτική χρονοπρογραμματισμού απλά ελέγχει το σύνολο των διεργασιών** που είναι διαθέσιμες σε ένα κόμβο και επιλέγει να εκτελεστεί η πιο κατάλληλη διεργασία με στόχο την μεγιστοποίηση της συνολικής απόδοσης.
- **Η πολιτική χρονοπρογραμματισμού των διεργασιών** οποιουδήποτε παραδοσιακού λειτουργικού συστήματος μπορεί να θεωρηθεί σαν πολιτική χρονοπρογραμματισμού, παρ' όλο που σε μερικά ειδικά συστήματα μπορούν να χρησιμοποιηθούν ειδικές τεχνικές, όπως για παράδειγμα η τεχνική *του co-scheduling*.

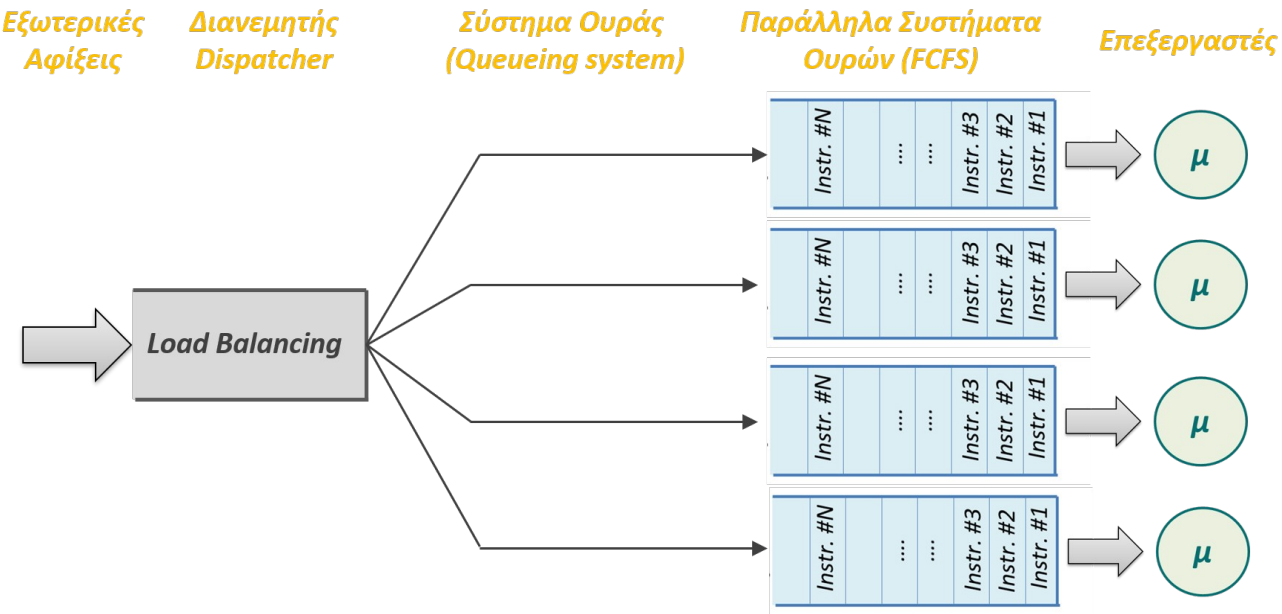
Στην πραγματικότητα, ο χρονοπρογραμματισμός επιτυγχάνεται κυρίως από το λειτουργικό σύστημα.

Πολιτικές ανάθεσης των tasks

Σε ένα κατανεμημένο σύστημα οι εργασίες (jobs) που φτάνουν σε αυτό, και πιο συγκεκριμένα τα παράλληλα tasks από τα οποία αποτελείται κάθε εργασία, πρέπει να εκχωρηθούν στους κόμβους του συστήματος για επεξεργασία. Όπως αναφέραμε και προηγουμένως, ο κανόνας που ακολουθείται για την εκχώρηση των εργασιών είναι γνωστός και ως **πολιτική ανάθεσης των tasks** (Task Allocation Policy).

Η επιλογή της πολιτικής ανάθεσης των εργασιών έχει **σημαντική επίδραση στην απόδοση** που παρατηρείται από τον χρήστη ενός τέτοιου συστήματος. Έτσι, ο σχεδιασμός ενός κατανεμημένου συστήματος καταλήγει στην επιλογή της καλύτερης δυνατής πολιτικής εκχώρησης των εργασιών για το συγκεκριμένο μοντέλο και τις απαιτήσεις των χρηστών.

Παρ' όλο που η επινόηση νέων πολιτικών είναι εύκολη, η ανάλυση ακόμα και των πιο απλών πολιτικών μπορεί να αποδειχθεί πολύ δύσκολη: Μερικές από τις πιο μακροχρόνιες αναπάντητες ερωτήσεις στην θεωρία των ουρών περιλαμβάνουν την ανάλυση της απόδοσης των πολιτικών ανάθεσης των εργασιών.



Εικόνα 1 Ένα τυπικό κατανεμημένο σύστημα με τέσσερις κόμβους

Στη συνέχεια περιγράφονται οι σημαντικότερες πολιτικές ανάθεσης των tasks που χρησιμοποιούνται πιο συχνά καθώς και μερικές ακόμα πολιτικές που έχουν προταθεί στην βιβλιογραφία. Σε όλες αυτές τις πολιτικές, η εκχώρηση των tasks στους κόμβους του συστήματος γίνεται μέσω **ενός κεντρικού διανεμητή (dispatcher)**, ο οποίος στις περισσότερες περιπτώσεις ***δεν γνωρίζει εκ των προτέρων*** το μέγεθος του κάθε task.

Η Εικόνα 1 δείχνει ένα τυπικό παράδειγμα ενός κατανεμημένου συστήματος το οποίο αποτελείται από τέσσερις κόμβους και έναν διανεμητή.

Στην πράξη μπορεί να λείπει η "μονάδα διανομής" και οι χρήστες του συστήματος μπορεί να αποφασίζουν από μόνοι τους σε ποιον κόμβο επιθυμούν να τρέξουν την εργασία τους.

Οι εργασίες που φτάνουν σε έναν κόμβο εκτελούνται σύμφωνα με την πειθαρχία First-Come-First-Served (FCFS), σύμφωνα με την οποία οι εργασίες (και πιο συγκεκριμένα τα tasks από τα οποία αποτελούνται οι εργασίες) εκτελούνται με την σειρά που αφίχθηκαν.

Οι πολιτικές ανάθεσης των tasks μπορούν να ομαδοποιηθούν σε τρείς κατηγορίες:

- **α) ΣΤΑΤΙΚΕΣ: Οι στατικές πολιτικές ανάθεσης δεν χρησιμοποιούν δυναμικές πληροφορίες, όπως για παράδειγμα τον φόρτο στο σύστημα. Χρησιμοποιούν στατικές πληροφορίες που συνήθως αφορούν το σύστημα στο οποίο θα εφαρμοστούν και σε εκτιμήσεις για την κατανομή των tasks που πρόκειται να φτάσουν στο σύστημα.**

Ένα μεγάλο πλεονέκτημα των τεχνικών αυτών είναι ότι δεν εισάγουν καθόλου κόστος εκτέλεσης (run-time overhead). Επίσης, οι τεχνικές αυτές είναι πιο εύκολο να σχεδιαστούν και μπορούν να είναι εξαιρετικά αποδοτικές όταν εφαρμόζονται σε ομοιογενή κατανεμημένα συστήματα.

Παρ' όλα αυτά, η απόδοσή τους εξαρτάται από τους προβλεπόμενους χρόνους εκτέλεσης και έτσι μπορεί να μην είναι κατάλληλες για δυναμικά ή απρόβλεπτα περιβάλλοντα και εφαρμογές.

Υπάρχουν δύο ευρέως γνωστές στατικές πολιτικές δρομολόγησης των tasks:
- η Πιθανοκρατική (Probabilistic) δρομολόγηση και
- η δρομολόγηση Κυκλικής Εναλλαγής (Round-Robin), οι οποίες περιγράφονται στη συνέχεια.

- **β) ΔΥΝΑΜΙΚΕΣ: Οι δυναμικές πολιτικές ανάθεσης προσπαθούν να βελτιώσουν τις κλασικές στατικές μεθόδους λαμβάνοντας υπόψη τον φόρτο του συστήματος για πιο έξυπνη κατανομή των tasks στους διαφόρους κόμβους του συστήματος.**

Οι τεχνικές αυτές εισάγουν όμως κάποιο κόστος εκτέλεσης, το οποίο προκύπτει λόγω του ότι απαιτείται η συνεχής παρατήρηση της τρέχουσας κατάστασης του συστήματος στη διάρκεια του χρόνου. Συνήθως οι δυναμικές τεχνικές δεν είναι πολύ πολύπλοκες στον σχεδιασμό τους και δεν εισάγουν μεγάλο κόστος εκτέλεσης. Έτσι, οι δυναμικές πολιτικές εκχώρησης έχουν γενικά την δυνατότητα να αποδώσουν καλύτερα από τις στατικές.

Δυναμικοί αλγόριθμοι δρομολόγησης είναι:
- Shortest-Queue (SQ) και
- Least-Work-Remaining (LWR), οι οποίοι ανήκουν στην γενική κατηγορία των Least-Loaded-First (LLF) πολιτικών,
- Central-Queue, και άλλοι.

- **γ) ΒΑΣΙΣΜΈΝΕΣ ΣΤΟ ΜΕΓΕΘΟΣ ΤΩΝ TASKS (Size-Based): Σε αυτές τις πολιτικές ανάθεσης κάθε κόμβος εξυπηρετεί tasks των οποίων οι απαιτήσεις εξυπηρέτησης ανήκουν σε μια καθορισμένη κατηγορία.**

Οι size-based πολιτικές χωρίζουν την κατανομή του μεγέθους των tasks σε έναν αριθμό διαστημάτων και κάθε κόμβος του κατανεμημένου συστήματος σχεδιάζεται, έτσι ώστε να εκτελεί tasks των οποίων το μέγεθος αντιστοιχεί σε ένα τέτοιο διάστημα.

Τα εισερχόμενα tasks στη συνέχεια εκχωρούνται στατικά σε κάποιον εξυπηρέτη ανάλογα με το μέγεθός τους. Με τον τρόπο αυτό ομαδοποιούνται σε κάθε κόμβο tasks με παρόμοιο μέγεθος, με στόχο να αποκλειστούν οι περιπτώσεις όπου ένα μικρό task μένει “κολλημένο” πίσω από ένα σημαντικά μεγαλύτερο task.

Περιγραφή βασικών τεχνικών ανάθεσης tasks και εξισορρόπησης φορτίου

Α) ΣΤΑΤΙΚΕΣ ΠΟΛΙΤΚΕΣ ΑΝΑΘΕΣΗΣ

Πιθανοκρατική δρομολόγηση

Σύμφωνα με την Πιθανοκρατική (Probabilistic) πολιτική ανάθεσης των tasks, κάθε εργασία, και πιο συγκεκριμένα κάθε παράλληλο task της εργασίας, αποστέλλεται στον κόμβο i πιθανότητα 1 / h, όπου h είναι ο συνολικός αριθμός των κόμβων του κατανεμημένου συστήματος.

Η πολιτική αυτή εξισώνει τον αναμενόμενο αριθμό των εργασιών σε κάθε κόμβο και χρησιμοποιείται συνήθως ως βάση για την σύγκριση άλλων πολιτικών δρομολόγησης.

Round-Robin δρομολόγηση

Η πολιτική κυκλικής εναλλαγής είναι η πιο συνηθισμένη πολιτική εκχώρησης των tasks, σύμφωνα με την οποία τα tasks ανατίθενται στους κόμβους του συστήματος με κυκλικό τρόπο με το ι-οστό task να εκχωρείται στον κόμβο i mod h, όπου h είναι ο συνολικός αριθμός των κόμβων του κατανεμημένου συστήματος.

Η πολιτική αυτή, όπως και η Πιθανοκρατική, είναι απλή και εξισώνει τον αναμενόμενο αριθμό των tasks σε κάθε κόμβο.

Β) ΔΥΝΑΜΙΚΕΣ ΠΟΛΙΤΚΕΣ ΑΝΑΘΕΣΗΣ

Least-Loaded-First δρομολόγηση

Σύμφωνα με την πολιτική LLF τα tasks εκχωρούνται στους εξυπηρέτες με τον λιγότερο υπολειπόμενο χρόνο εργασίας, προσπαθώντας έτσι να επιτύχουν μια στιγμιαία εξισορρόπηση του φόρτου. Ο υπολειπόμενος χρόνος εργασίας σε κάθε εξυπηρέτη μπορεί να μετρηθεί με δύο τρόπους:

α) Shortest Queue LLF: Στην πρώτη περίπτωση θεωρείται ότι δεν είναι γνωστό το μέγεθος των tasks και ο υπολειπόμενος χρόνος εργασίας μπορεί να μετρηθεί προσεγγιστικά

από το μήκος της ουράς κάθε εξυπηρέτη, δηλαδή από τον αριθμό των tasks που περιμένουν για εξυπηρέτηση στην ουρά του εξυπηρέτη. Η πολιτική αυτή ονομάζεται Shortest Queue και τα εισερχόμενα tasks δρομολογούνται στον εξυπηρέτη με τον μικρότερο αριθμό tasks.

β) Least-Work-Remaining (LWR): Στη δεύτερη περίπτωση θεωρείται ότι το μέγεθος των tasks είναι γνωστό εκ των προτέρων και κάθε εισερχόμενο task εκχωρείται στον εξυπηρέτη με το μικρότερο άθροισμα των μεγεθών των εργασιών που περιμένουν στην ουρά και του υπολειπόμενου χρόνου του task που εξυπηρετείται. Η πολιτική αυτή ονομάζεται Least-Work-Remaining (LWR) και πλησιάζει περισσότερο στην επίτευξη εξισορροπημένου φόρτου στο σύστημα, απαιτεί όμως την επιπλέον γνώση των απαιτήσεων εξυπηρέτησης των tasks που φτάνουν στο σύστημα και η οποία δεν είναι πάντα διαθέσιμη.

Στις περιπτώσεις όπου η LLF βασίζεται στον αριθμό των εργασιών στις ουρές για την εξισορρόπηση των εισερχόμενων tasks και όπου η μεταβλητότητα του μεγέθους των tasks είναι υψηλή, είναι προφανές ότι η πληροφορία αυτή δεν είναι αντιπροσωπευτική του φόρτου σε κάθε εξυπηρέτη και αυτό επιδρά αρνητικά στην απόδοση του συστήματος.

Central-Queue δρομολόγηση

Η πολιτική Central-Queue, κρατάει όλα τα tasks στην μονάδα του διανεμητή σε μια ουρά με πειθαρχία εξυπηρέτησης FCFS, και μόνο όταν ένας κόμβος του συστήματος είναι ελεύθερος ζητάει κάποιο task από τον διανεμητή και λαμβάνει έτσι το επόμενο task στην ουρά.

Η πολιτική αυτή μεγιστοποιεί την χρησιμοποίηση των κόμβων και με τον τρόπο αυτό βοηθάει στην ελαχιστοποίηση του μέσου χρόνου απόκρισης.

Γ) ΠΟΛΙΤΚΕΣ ΑΝΑΘΕΣΗΣ ΒΑΣΙΣΜΕΝΕΣ ΣΤΟ ΜΕΓΕΘΟΣ ΤΩΝ TASKS

Πολιτικές ανάθεσης που βασίζονται σε μεγέθη

Όπως αναφέραμε προηγουμένως, στις size-based πολιτικές κάθε κόμβος του συστήματος σχετίζεται με ένα εύρος μεγέθους και κάθε task αποστέλλεται στον κατάλληλο κόμβο ανάλογα με το μέγεθος του.

Code (Python):

```
import random
import matplotlib.pyplot as plt

class Processor:
    def __init__(self, serviceRate, queueSize, name):
        self.serviceRate = serviceRate
        self.queueSize = queueSize
        self.name = name
        self.queue = []
        self.served = 0
        self.dropped = 0
        self.busyUntil = 0
        self.waitingTime = 0
        self.wasFree = True
        self.totalBusyTime = 0


    def isFree(self, currentTime):
        return self.busyUntil <= currentTime


    def addTask(self, currentTime):
        if len(self.queue) < self.queueSize:
            self.queue.append(currentTime)
        else:
            self.dropped += 1


    def tick(self, currentTime):
        served = False

        if self.isFree(currentTime):
            if not self.wasFree:
                self.served += 1
                self.waitingTime += currentTime - self.queue.pop(0)
                self.wasFree = True
                served = True

            if self.queue:
                self.busyUntil = currentTime + random.expovariate(self.serviceRate)
                self.totalBusyTime += self.busyUntil - currentTime
                self.wasFree = False

        return served


    def getStatus(self):
        status = {
            "name": self.name,
            "total": self.served + self.dropped + len(self.queue),
            "served": self.served,
            "dropped": self.dropped,
            "queue": len(self.queue),
            "waitingTime": self.waitingTime,
            "totalBusyTime": self.totalBusyTime
        }

        return status
```

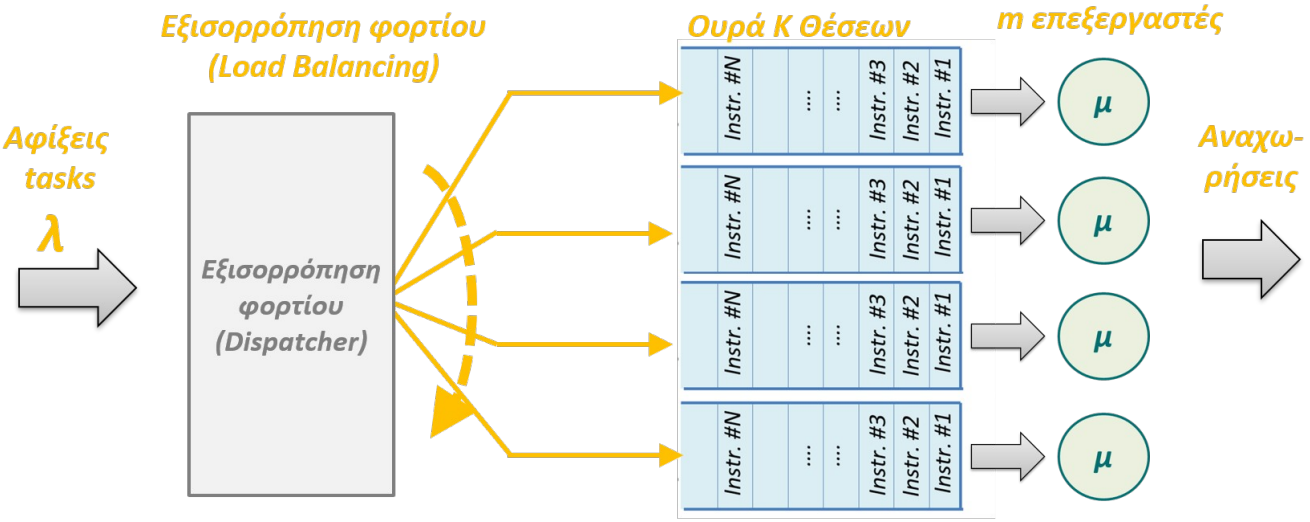
```
class LoadBalancer:
  def __init__(self, arrivalRate, processors, simulationTime):
    self.arrivalRate = arrivalRate
    self.processors = []
    self.processorWeights = []
    self.simulationTime = simulationTime
    self.assignmentFunc = None
    self.assignmentVars = {}
    self.arrivalsInTime = {}
    self.departuresInTime = {}

    for i, processor in enumerate(processors):
      self.processors.append(Processor(processor[0], processor[1], i))

      if len(processor) == 2:
        self.assignmentFunc = self.roundRobin

      elif len(processor) > 2:
        self.processorWeights.append(processor[2])
        self.assignmentFunc = self.weightedRoundRobin

    self.start()


  def start(self):
    currentTime = random.expovariate(self.arrivalRate)

    while currentTime < self.simulationTime:
      self.arrivalsInTime[currentTime] = self.arrivalsInTime.get("last", 0) + 1
      self.arrivalsInTime["last"] = self.arrivalsInTime[currentTime]

      nextProcessor = self.assignmentFunc()
      self.processors[nextProcessor].addTask(currentTime)

      for processor in self.processors:
        if processor.tick(currentTime):
          self.departuresInTime[currentTime] = self.departuresInTime.get("last", 0) + 1
          self.departuresInTime["last"] = self.departuresInTime[currentTime]

      currentTime += random.expovariate(self.arrivalRate)

    del self.arrivalsInTime["last"]
    del self.departuresInTime["last"]

    self.stats()
    self.plotArrivals()


  def stats(self):
    totalTasks = 0
    totalServed = 0
    totalDropped = 0
    totalLeftInQueue = 0
    totalWaitingTime = 0

    for processor in self.processors:
      status = processor.getStatus()

      totalTasks += status["total"]
      totalServed += status["served"]
      totalDropped += status["dropped"]
      totalLeftInQueue += status["queue"]
```

```
            totalWaitingTime += status["waitingTime"]

            print(f"Processor {status["name"]}:")
            print(f"    Total tasks: {status["total"]}")
            print(f"    Served tasks: {status["served"]}")
            print(f"    Dropped tasks: {status["dropped"]} ({round(status["dropped"] / status["total"] * 100, 2)}%)")
            print(f"    Tasks left in queue: {status["queue"]}")
            print(f"    Average waiting time: {round(status["waitingTime"] / status["served"], 2)}")
            print(f"    Total busy time: {round(status["totalBusyTime"] / self.simulationTime * 100, 2)}%")
            print("\n\n")

        print(f"Total tasks: {totalTasks}")
        print(f"Total served: {totalServed}")
        print(f"Total dropped: {totalDropped} ({round(totalDropped / totalTasks * 100, 2)}%)")
        print(f"Total left in queue: {totalLeftInQueue}")
        print(f"Total average waiting time: {round(totalWaitingTime / totalServed, 2)}")


    def roundRobin(self):
        if not self.assignmentVars:
            self.assignmentVars["next"] = 0

            return self.assignmentVars["next"]

        self.assignmentVars["next"] = (self.assignmentVars["next"] + 1) % len(self.processors)

        return self.assignmentVars["next"]


    def weightedRoundRobin(self):
        if not self.assignmentVars:
            for i in range(len(self.processors)):
                self.assignmentVars[i] = 0

            self.assignmentVars["next"] = 0

        if self.assignmentVars[self.assignmentVars["next"]] < self.processorWeights[self.assignmentVars["next"]]:
            self.assignmentVars[self.assignmentVars["next"]] += 1

            return self.assignmentVars["next"]

        self.assignmentVars[self.assignmentVars["next"]] = 0
        self.assignmentVars["next"] = (self.assignmentVars["next"] + 1) % len(self.processors)
        self.assignmentVars[self.assignmentVars["next"]] += 1

        return self.assignmentVars["next"]


    def plotArrivals(self):
        plt.plot(list(self.arrivalsInTime.keys()), list(self.arrivalsInTime.values()), label="Arrivals")
        plt.plot(list(self.departuresInTime.keys()), list(self.departuresInTime.values()), label="Departures")
        plt.legend()
        plt.show()

# Παράδειγμα κώδικα εκτέλεσης για 4 επεξεργαστές με ρυθμό επεξεργασίας μ = 1 και ουρές 10 θέσεων
if __name__ == "__main__":
    processors = [(1, 10), (1, 10), (1, 10), (1, 10)]

    LoadBalancer(5, processors, 1000)
```

1η σειρά από ερωτήματα για συμμετρικό σύστημα τεσσάρων παράλληλων ισάξιων επεξεργαστών με ανάθεση κατά Round Robin :

Στην παρούσα άσκηση και σειρά από ερωτήματα θα μελετηθεί ένα συμμετρικό σύστημα τεσσάρων παράλληλων και ισάξιων επεξεργαστών με τη δική τους ουρά ο καθένας, όπως απεικονίζεται στο παρακάτω σύστημα. Να προσομοιώσετε το παρακάτω σύστημα για τις εξής τιμές:

- Άφιξη εργασιών με εκθετική κατανομή και ρυθμό άφιξης λ=5
- Τέσσερις παράλληλοι επεξεργαστές με εκθετική κατανομή και ρυθμό επεξεργασίας μ = 1
- Κοινή ουρά με K= 10 θέσεις
- Επιλογή επεξεργαστή εκ-περιτροπής, δηλαδή μία εργασία στον καθένα και κυκλικά και στους τέσσερις
- Χρόνος προσομοίωσης t = 1000



Εικόνα 2 Σύστημα τεσσάρων επεξεργαστών με επιλογή κατά round robin

1) Να συμπεριλάβετε εδώ τον κώδικά σας για την παραπάνω προσομοίωση και να παρουσιάσετε τα αποτελέσματα για

a. Το μέσο χρόνο αναμονής στην ουρά των tasks.
b. Τον συνολικό-αθροιστικό αριθμό αφίξεων νέων tasks στο σύστημα.
c. Το συνολικό-αθροιστικό αριθμό αναχωρήσεων-εξυπηρετήσεων των tasks στο σύστημα.
d. Το πλήθος των εργασιών που έχουν εξυπηρετηθεί από τον κάθε ένα επεξεργαστή και συνολικά από το σύστημα
d. Το μέσο συντελεστή χρησιμοποίησης του κάθε επεξεργαστή - ποσοστό utilization.
e. Το πλήθος των tasks που έχουν γίνει dropped

ΑΠΑΝΤΗΣΗ:

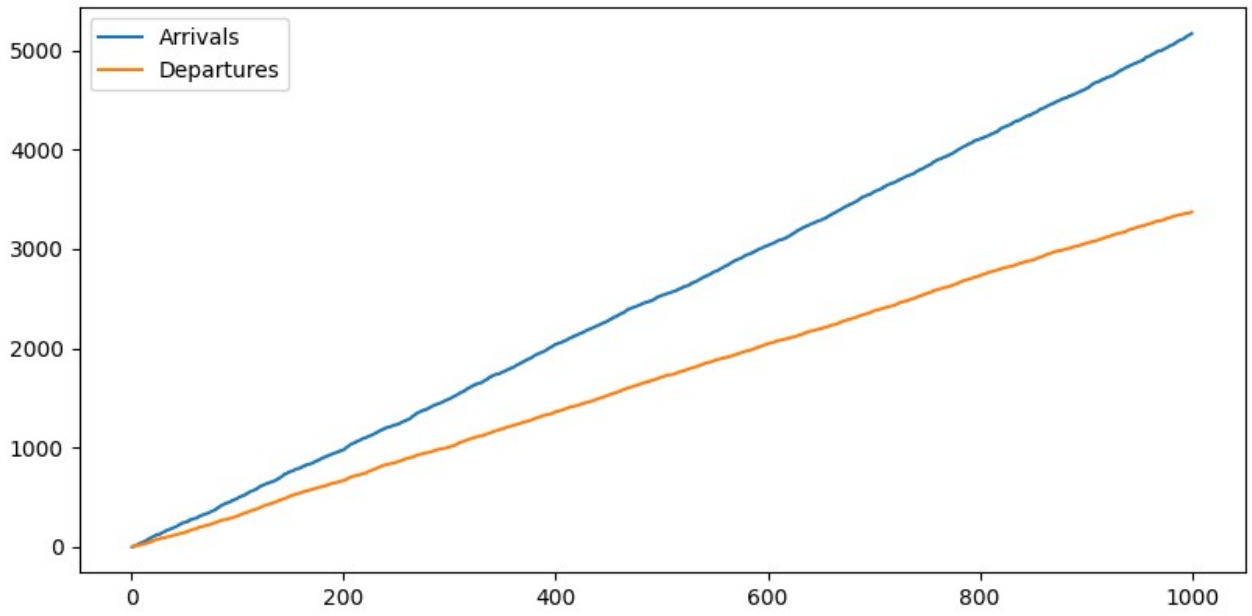
Κώδικας εκτέλεσης (μπαίνει κάτω κάτω στο πρόγραμμα):

```
if __name__ == "__main__":
    processors = [
        (1, 10),
        (1, 10),
        (1, 10),
        (1, 10)
    ]

    LoadBalancer(5, processors, 1000)
```

```
Processor 0:
    Total tasks: 1292
    Served tasks: 877
    Dropped tasks: 407 (31.5%)
    Tasks left in queue: 8
    Average waiting time: 10.23
    Total busy time: 82.88%


Processor 1:
    Total tasks: 1292
    Served tasks: 833
    Dropped tasks: 449 (34.75%)
    Tasks left in queue: 10
    Average waiting time: 10.62
    Total busy time: 84.65%
```

```
Processor 2:
    Total tasks: 1292
    Served tasks: 822
    Dropped tasks: 461 (35.68%)
    Tasks left in queue: 9
    Average waiting time: 11.16
    Total busy time: 85.06%


Processor 3:
    Total tasks: 1292
    Served tasks: 840
    Dropped tasks: 443 (34.29%)
    Tasks left in queue: 9
    Average waiting time: 10.79
    Total busy time: 83.34%
```

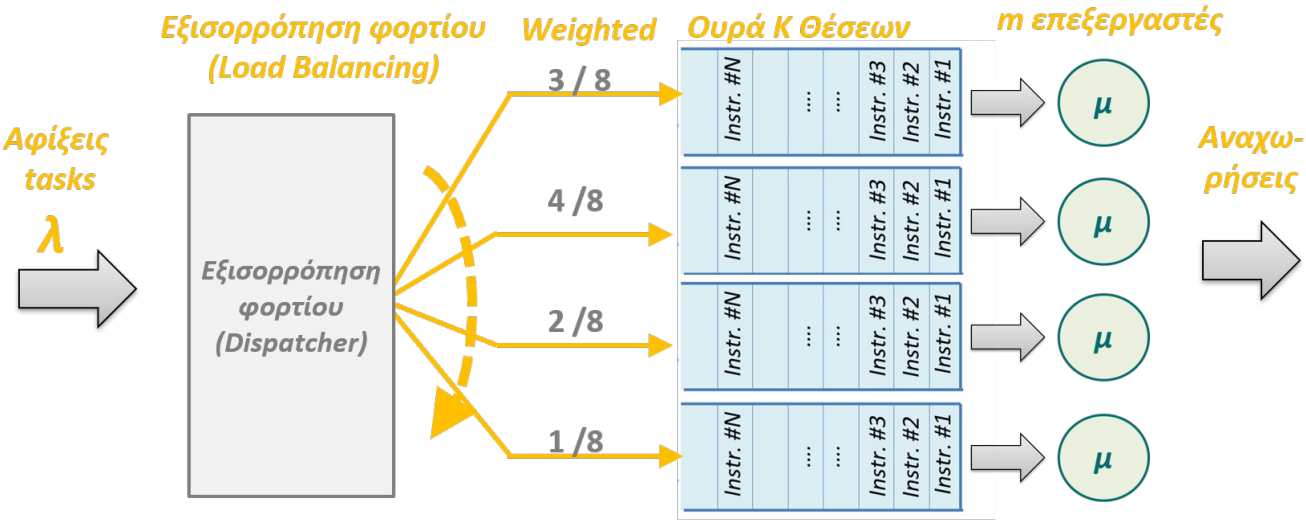
```
Total tasks: 5168
Total served: 3372
Total dropped: 1760 (34.06%)
Total left in queue: 36
Total average waiting time: 10.69
```



2η σειρά από ερωτήματα για συμμετρικό σύστημα τεσσάρων παράλληλων ισάξιων επεξεργαστών με ανάθεση κατά Weighted Round Robin :

Να μελετηθεί ένα συμμετρικό σύστημα τεσσάρων παράλληλων και ισάξιων επεξεργαστών με τη δική τους ουρά ο καθένας, όπως απεικονίζεται στο παρακάτω σύστημα αλλά με πολιτική ανάθεσης weighted round robin και τις παρακάτω τιμές:

- Άφιξη εργασιών με εκθετική κατανομή και ρυθμό άφιξης λ=5
- Τέσσερις παράλληλοι επεξεργαστές με εκθετική κατανομή και ρυθμό επεξεργασίας μ = 1
- Κοινή ουρά με K= 10 θέσεις
- Επιλογή επεξεργαστή εκ-περιτροπής με βάρη, όπως αυτά απεικονίζονται στην εικόνα, δηλαδή τρεις, τέσσερις, δύο και μία διαδοχικές εργασίες να ανατίθενται στον επεξεργαστή ένα, δύο, τρία και τέσσερα αντίστοιχα και έπειτα κυκλικά ξανά και στους τέσσερις
- Χρόνος προσομοίωσης t = 1000



Εικόνα 3 Σύστημα τεσσάρων επεξεργαστών με επιλογή κατά weighted round robin

2) Να συμπεριλάβετε εδώ τον κώδικά σας για την παραπάνω προσομοίωση και να παρουσιάσετε τα αποτελέσματα για

a. Το μέσο χρόνο αναμονής στην ουρά των tasks.
b. Τον συνολικό-αθροιστικό αριθμό αφίξεων νέων tasks στο σύστημα.
c. Το συνολικό-αθροιστικό αριθμό αναχωρήσεων-εξυπηρετήσεων των tasks στο σύστημα.
d. Το πλήθος των εργασιών που έχουν εξυπηρετηθεί από τον κάθε ένα επεξεργαστή και συνολικά από το σύστημα
d. Το μέσο συντελεστή χρησιμοποίησης του κάθε επεξεργαστή - ποσοστό utilization.
e. Το πλήθος των tasks που έχουν γίνει dropped

ΑΠΑΝΤΗΣΗ:

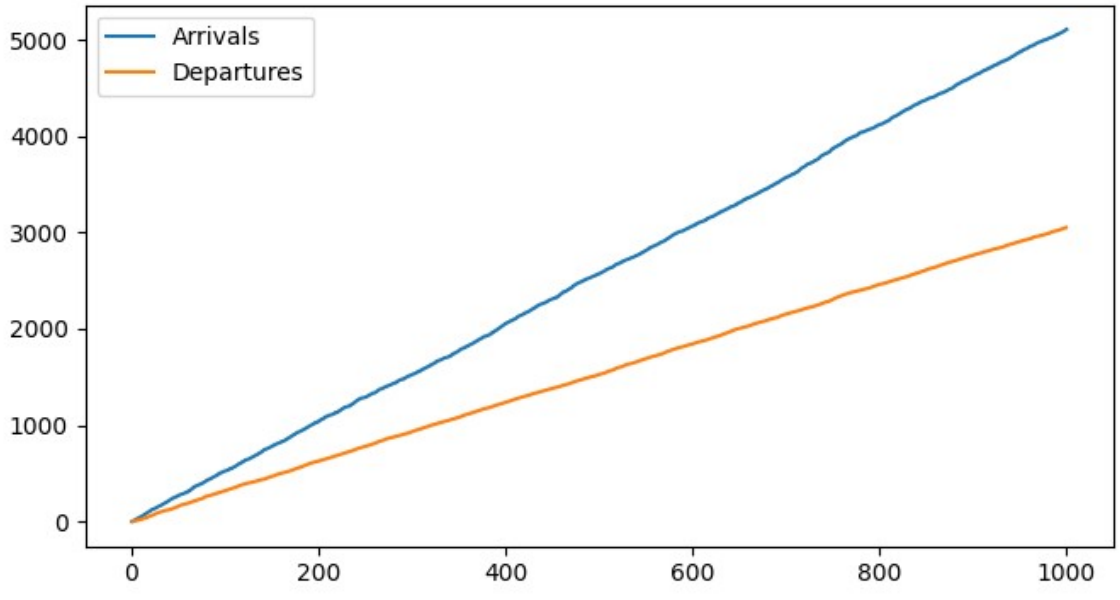
Κώδικας εκτέλεσης (μπαίνει κάτω κάτω στο πρόγραμμα):

```
if __name__ == "__main__":
    processors = [
        (1, 10, 3),
        (1, 10, 4),
        (1, 10, 2),
        (1, 10, 1)
    ]

    LoadBalancer(5, processors, 1000)
```

```
Processor 0:
    Total tasks: 1533
    Served tasks: 865
    Dropped tasks: 658 (42.92%)
    Tasks left in queue: 10
    Average waiting time: 10.55
    Total busy time: 83.74%


Processor 1:
    Total tasks: 2044
    Served tasks: 864
    Dropped tasks: 1170 (57.24%)
    Tasks left in queue: 10
    Average waiting time: 10.79
    Total busy time: 83.58%
```

```
Processor 2:
    Total tasks: 1021
    Served tasks: 812
    Dropped tasks: 199 (19.49%)
    Tasks left in queue: 10
    Average waiting time: 9.81
    Total busy time: 83.8%


Processor 3:
    Total tasks: 510
    Served tasks: 510
    Dropped tasks: 0 (0.0%)
    Tasks left in queue: 0
    Average waiting time: 1.61
    Total busy time: 50.5%
```

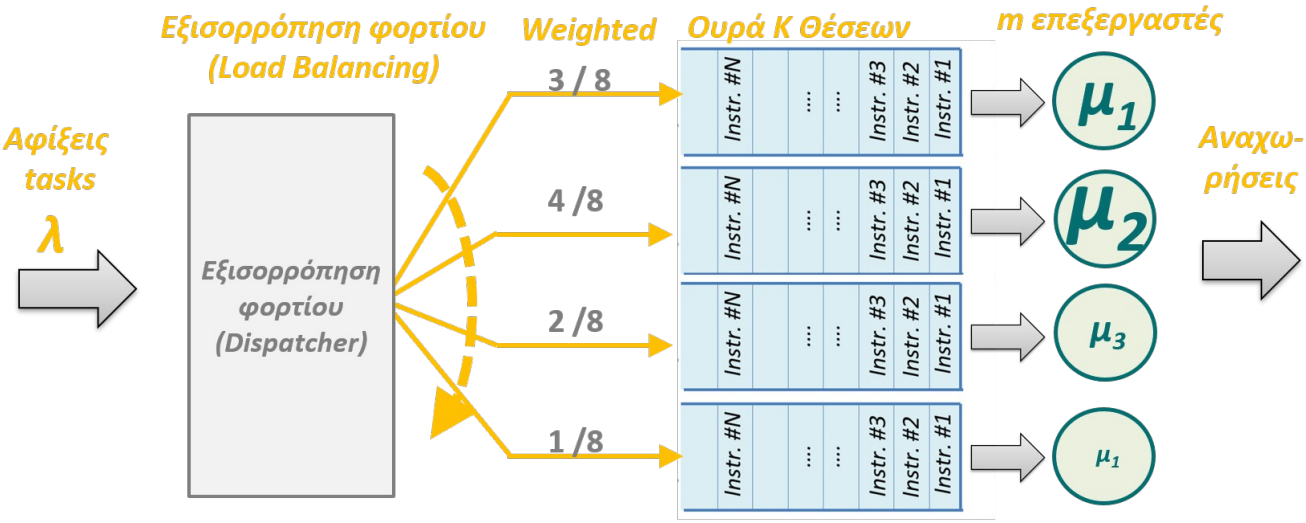
```
Total tasks: 5108
Total served: 3051
Total dropped: 2027 (39.68%)
Total left in queue: 30
Total average waiting time: 8.93
```



3η σειρά από ερωτήματα για ασύμμετρο σύστημα τεσσάρων παράλληλων άνισων επεξεργαστών με ανάθεση κατά Weighted Round Robin :

Να μελετηθεί το παρακάτω ασύμμετρο σύστημα τεσσάρων παράλληλων επεξεργαστών με δική τους ουρά ο καθένας και ρυθμούς επεξεργασίας $\mu_1 = 5$, $\mu_2 = 7$, $\mu_3 = 4$, $\mu_1 = 2$. Η πολιτική ανάθεσης weighted round robin και η προσομοίωση να ακολουθεί τις παρακάτω τιμές:

- Άφιξη εργασιών με εκθετική κατανομή και ρυθμό άφιξης λ=5
- Τέσσερις παράλληλοι επεξεργαστές με εκθετική κατανομή και ρυθμό επεξεργασίας μ = 1
- Κοινή ουρά με K= 10 θέσεις
- Επιλογή επεξεργαστή εκ-περιτροπής με βάρη, όπως αυτά απεικονίζονται στην εικόνα, δηλαδή τρεις, τέσσερις, δύο και μία διαδοχικές εργασίες να ανατίθενται στον επεξεργαστή ένα, δύο, τρία και τέσσερα αντίστοιχα και έπειτα κυκλικά ξανά και στους τέσσερις
- Χρόνος προσομοίωσης t = 1000



Εικόνα 4 Σύστημα τεσσάρων άνισων επεξεργαστών με επιλογή κατά weighted round robin

3) Να συμπεριλάβετε εδώ τον κώδικά σας για την παραπάνω προσομοίωση και να παρουσιάσετε τα αποτελέσματα για

a. Το μέσο χρόνο αναμονής στην ουρά των tasks.
b. Τον συνολικό-αθροιστικό αριθμό αφίξεων νέων tasks στο σύστημα.
c. Το συνολικό-αθροιστικό αριθμό αναχωρήσεων-εξυπηρετήσεων των tasks στο σύστημα.
d. Το πλήθος των εργασιών που έχουν εξυπηρετηθεί από τον κάθε ένα επεξεργαστή και συνολικά από το σύστημα
d. Το μέσο συντελεστή χρησιμοποίησης του κάθε επεξεργαστή - ποσοστό utilization.
e. Το πλήθος των tasks που έχουν γίνει dropped

4) Για τους συγκεκριμένους αριθμούς επεξεργασίας, ποιοι συντελεστές βαρών θα απέδιδαν πιο δίκαιη και συμμετρική απόδοση τους συστήματος; Να επαναλάβετε την προσομοίωση για τους συντελεστές αυτούς

ΑΠΑΝΤΗΣΗ:

Κώδικας εκτέλεσης (μπαίνει κάτω κάτω στο πρόγραμμα):

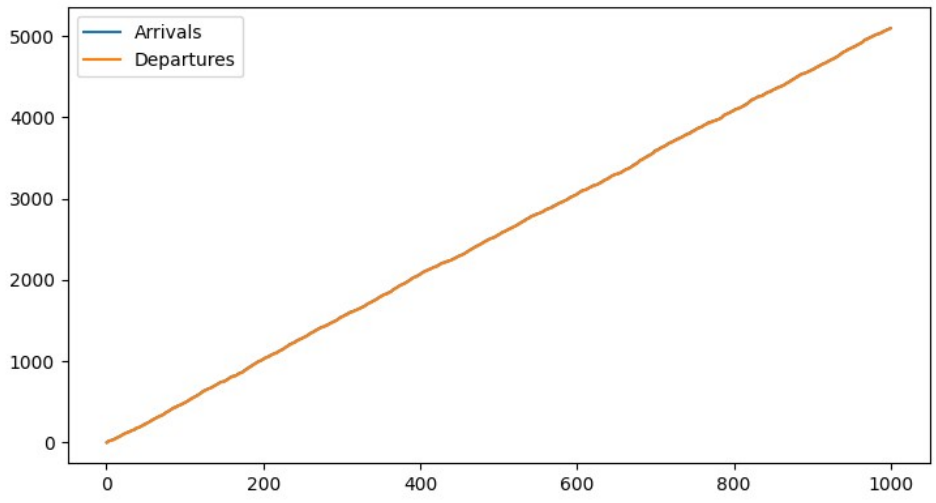
```
if __name__ == "__main__":
    processors = [
        (5, 10, 3),
        (7, 10, 4),
        (4, 10, 2),
        (2, 10, 1)
    ]

    LoadBalancer(5, processors, 1000)
```

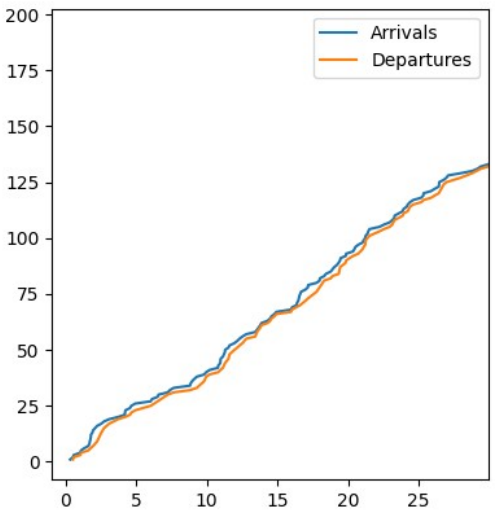
```
Processor 0:
    Total tasks: 1530
    Served tasks: 1529
    Dropped tasks: 0 (0.0%)
    Tasks left in queue: 1
    Average waiting time: 0.64
    Total busy time: 32.07%

Processor 1:
    Total tasks: 2040
    Served tasks: 2039
    Dropped tasks: 0 (0.0%)
    Tasks left in queue: 1
    Average waiting time: 0.59
    Total busy time: 29.34%
```

```
Processor 2:
    Total tasks: 1018
    Served tasks: 1018
    Dropped tasks: 0 (0.0%)
    Tasks left in queue: 0
    Average waiting time: 0.6
    Total busy time: 25.36%

Processor 3:
    Total tasks: 509
    Served tasks: 509
    Dropped tasks: 0 (0.0%)
    Tasks left in queue: 0
    Average waiting time: 0.7
    Total busy time: 25.79%
```

```
Total tasks: 5097
Total served: 5095
Total dropped: 0 (0.0%)
Total left in queue: 2
Total average waiting time: 0.62
```

Ο ίδιος πίνακας αλλά με μεγέθυνση





ΠΑΡΑΔΟΣΗ ΕΡΓΑΣΙΑΣ

Παράδοση εργαστηριακής άσκησης πάντα μόνο μέσω eclass και με χρήση του παρόντος απαντητικού φύλλου

Πηγές:

[1] Διαλέξεις των προπτυχιακών και μεταπτυχιακών μαθημάτων *«Απόδοση Παράλληλων Συστημάτων»* και *«Μοντελοποίηση και Προσομοίωση*, ακαδημαϊκά έτη 2010-2012, του Τμ. Πληροφορικής Α.Π.Θ. της καθηγ. Ελένης Καρατζά.

[2] *"Ανάλυση Επίδοσης Υπολογιστικών Συστημάτων: Αναλυτικά μποντέλα, προσομοίωση, μετρήσεις*", Α.Γ. Σταφυλοπάτης, Γ. Σιόλας, Ελληνικά Ακαδημαϊκά Ηλεκτρονικά Συγγραματα και Βοηθήματα, 2015, της Δράσης των Ανοικτών Ακαδημαϊκών Ηλεκτρονικών Συγγραμμάτων - Αποθετήριο Κάλλιπος, Υπερ-σύνδεσμος https://repository.kallipos.gr/handle/11419/6055