Εθνικο Μετσοβιο Πολυτεχνειο

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΠΙΔΟΣΗ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

2η ΑΣΚΗΣΗ

Αχλάτης Στέφανος-Σταμάτης (03116149)

<el16149@central.ntua.gr>

Ηλιακοπούλου Νικολέτα-Μαρκέλα (03116111)

<el16111@central.ntua.gr>

Ιούνιος 2020

Εισαγωγή

Στην κάτωθι εργασία καλούμαστε να υλοποιήσουμε σε γλώσσα προγραμματισμού της επιλογής μας πρόγραμμα που προσομοιώνει την λειτουργία ενός οργανισμού παροχής υπηρεσιών ο οποίος υποστηρίζεται από έναν server ο περιλαμβάνει CPU και έναν Δίσκο.

Εργαλεία

Σαν γλώσσα προγραμματισμού επιλέχθηκε η Python3 μια γλώσσα πού συνηθισμένη για προγράμματα προσομοιώσεων λόγω της αντικειμενοστρέφειας της και της ευκολίας στην χρήση της.
Επίσης χρησιμοποιούμε πολλές βιβλιοθήκες της Python3 όπως η numpy για διαχείριση πινάκων και υπολογισμών, παραγωγή τυχαίων αριθμών και η scipy για υπολογισμό των τιμών της κατανομής Student.Η υλοποίηση της προσομοίωσης έγινε “με το χέρι”, χωρίς τη χρήση του simpy ή κάποιου αντίστοιχου εργαλείου.

Μέθοδος

Οι εργασίες στο σύστημα φτάνουν μέσω διαδικτύου με ρυθμό αφίξεων που ακολουθεί την κατανομή Poisson με ρυθμο 2.4 εργασίες ανα δευτερόλεπτο.
Πιο αναλυτικά το 70% των αιτήσεων προέρχεται από εγγεγραμμένους χρήστες, που καλούνται subscribers ενώ το υπόλοιπο 30% προέρχεται από επισκέπτες, δηλαδή χρήστες που δεν έχουν προφίλ στον οργανισμό.
Σύμφωνα με τον κανόνα της **διάσπασης της poisson** μπορεί να θεωρηθεί ότι στο σύστημα έρχονται 0.7*2.4 δηλαδή 1.68 εργασίες από subscribers ανά δευτερόλεπτο και 0.3*2.4 δηλαδή 0.72 εργασίες από visitors ανά δευτερόλεπτο.
Η επεξεργασία μιας εργασίας οποιασδήποτε κατηγορίας στη CPU διακόπτεται όποτε χρειάζεται προσπέλαση στον δίσκο. Μετά την εξυπηρέτηση στον δίσκο, η εργασία επιστρέφει στη CPU, όπου συνεχίζεται η επεξεργασία της. Οταν περατωθεί η εκτέλεση μιας εργασίας, το αποτέλεσμα μεταδίδεται προς το Διαδίκτυο μέσω εξερχόμενης σύνδεσης.
Ένας ακόμη λόγος που δεν μπορούμε να χρησιμοποιήσουμε το **εργαλείο JMT** είναι επειδή οι χρήστες χαρακτηρίζονται από ένα χαρακτηριστικό ανυπομονησίας, λειτουργία που δεν παρέχεται στο JMT.
Συγκεκριμένα, κάθε εργασία της κατηγορίας subscriber, κατά την άφιξή της στο σύστημα, «λαμβάνει γνώση» του συνολικού αριθμού k των εργασιών των δύο κατηγοριών που βρίσκονται τη στιγμή εκείνη στη CPU και στον δίσκο. Με βάση την πληροφορία αυτή, η εργασία «αποφασίζει» αν θα προχωρήσει κανονικά ή θα «ματαιωθεί», προκειμένου να αποφύγει ενδεχόμενη μεγάλη καθυστέρηση. Συνεπώς, κάθε εργασία κατηγορίας subscriber, που φθάνει στο σύστημα, διαθέτει ένα ανώτατο όριο ξ, τέτοιο ώστε να σημειώνεται οπισθοχώρηση όταν k > ξ. Σύμφωνα με μετρήσεις, το όριο ξ κάθε εργασίας της κατηγορίας subscriber μπορεί να παρασταθεί ως τυχαία μεταβλητή που ακολουθεί κανονική κατανομή με μέση τιμή μ=40,5 και τυπική απόκλιση σ=6,0. Όσον αφορά την κατηγορία visitor, έχει παρατηρηθεί ότι οι χρήστες διακόπτουν την παραμονή τους στο σύστημα όταν ο χρόνος απόκρισης υπερβαίνει έναν χρόνο κατωφλίου, που υποδηλώνει πόσο είναι διατεθειμένος ο χρήστης να αναμείνει την απάντηση. Εκτιμάται ότι, για κάθε εργασία, ο χρόνος κατωφλίου ακολουθεί την **κατανομή Pareto** (Τύπου I) με συνάρτηση κατανομής πιθανότητας:

$$F(x) = 1 - \left(\frac{xm}{x}\right)^{a} \quad x \geq x_m, x_m = 4.2\ ,\ a = 1.5$$

Δηλαδή,

$$F(x) = 1 - \left(\frac{4.2}{x}\right)^{1.5} \quad , x \geq 4.2$$

Και στην συνέχεια κάντας χρήση της **μεθόδου της αντιστροφής** υπολογίζουμε την αντιστροφη: $X = F^{-}(U)$ που βρίσκουμε ότι είναι η εξής:

$$X = \sqrt[3]{\frac{74.088}{1-2U+U^2}}$$

Όπου U είναι μια γεννήτρια τυχαίων αριθμών μεταξύ 0 και 1, ομοιόμορφα κατανεμημένων.

Παρακάτω παρατίθεται και η σχετική υλοποίηση στην προσομοίωση:

```
def my_paretovariate():
    '''
    Sample Pareto variate with xm=4.2 , a=1.5
    '''
    U = random.uniform(0,1)
    temp2 = 74.088/(1 - 2*U + U*U)
    final = temp2 ** (1. / 3)
  #  print (f'final is {final}')
    return final
```

Οι ακόλουθες πληροφορίες προέρχονται από μετρήσεις στο σύστημα. Οι χρόνοι αναφέρονται σε msec και αφορούν μέσους χρόνους ανά επίσκεψη.

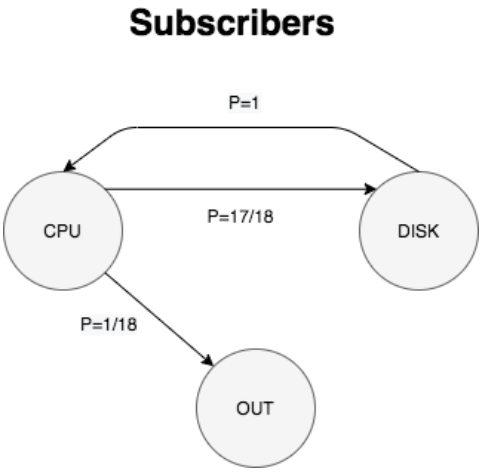
	subscriber	visitor
Μέσος χρόνος εξυπηρέτησης στη CPU	24	21
Μέσος χρόνος εξυπηρέτησης στον δίσκο	25	30
Μέσος αριθμός επισκέψεων στον δίσκο	17	9
Μέσος χρόνος μετάδοσης αποτελέσματος (εξερχόμενη σύνδεση)	436	321

Θα υποθέσουμε ότι ο χρόνος εξυπηρέτησης είναι κατανεμημένος εκθετικά για τον δίσκο και την εξερχόμενη σύνδεση, και ακολουθεί Erlang-4 στη CPU. Με βάση τον μέσο αριθμό επισκέψεων προσδιορίζονται οι αντίστοιχες πιθανότητες δρομολόγησης. Υποθέτουμε ότι στη CPU οι εργασίες εξυπηρετούνται με κανονισμό Processor Sharing, ενώ στους δίσκους και στην εξερχόμενη σύνδεση με κανονισμό FIFO.

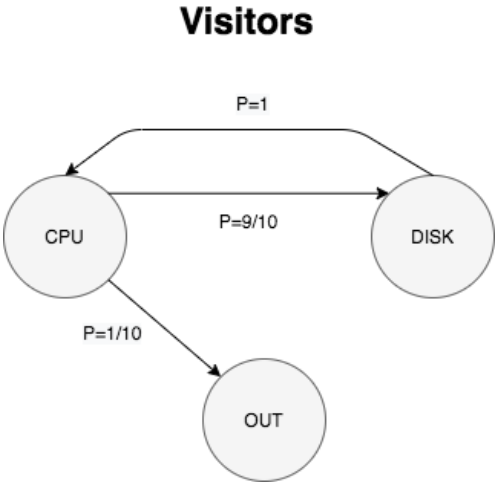
Θα χρησιμοποιηθεί η αναγεννητική μέθοδος με βαθμό εμπιστοσύνης 95%. Το διάστημα εμπιστοσύνης μπορεί να υπολογίζεται κάθε 20 αναγεννησιακούς κύκλους. Η εκτέλεση του προγράμματος θα σταματά όταν το διάστημα εμπιστοσύνης για τον μέσο χρόνο απόκρισης της κατηγορίας subscriber έχει μήκος μικρότερο από το 10% της μέσης τιμής ή όταν εκτελεστούν 1000 αναγεννητικοί κύκλοι.

Σημασιολογία τοπολογίας Δικτύου:

Με βάση τα προηγούμενα γίνεται σαφής η σημασιολογία της τοπολογίας του δικτυου ως εξής:
Για την κλάση των εργασιών τύπου subscriber:



Για την κλάση των εργασιών τύπου visitors:



Τεχνικά χαρακτηριστικά προσομοίωσης - Καταστάσεις συστήματος
Σύμφωνα με την προηγούμενη ανάλυση χαρακτηριστικά γεγονότα της προσομοίωσης είναι τα εξής:

- Άφιξη στο σύστημα, που συνεπάγεται με πιθανότητα 1 ότι πάει στην CPU
- Από CPU στον δίσκο
- Από δίσκο στην CPU
- Από CPU στον κόμβο OUT
- Από τον κόμβο OUT στην πλήρης αποχώρηση από το σύστημα μέσω της εξωτερικής σύνδεσης

Τα οποία μοντελοποιήθηκαν ως ξεχωριστές καταστάσεις του συστήματος **για κάθε κατηγορία** πελατών.
Αναγεννησιακοί κύκλοι και Σύγκλιση

Θεωρούμε ως αναγεννησιακό κύκλο κάθε στιγμή που ένας πελάτης αποχωρεί πλήρως από το σύστημα.
Για κάθε ολοκληρωμένο κύκλο κρατάμε την χρονική του διάρκεια T και τη λίστα με τους χρόνους εξυπηρέτησης R κατά τον κύκλο αυτό. Με βάση αυτά, μπορούμε να υπολογίσουμε το throughput του κύκλου ως:

$$X = \frac{\text{αριθμός πελατών που εξυπερετήθηκαν}}{\text{χρονική διάρκεια κύκλου}} = \frac{len(R)}{T}$$

Έπειτα, με βάση τα αντίστοιχα throughput, υπολογίζουμε τους αντίστοιχους βαθμούς χρησιμοποίησης για κάθε πόρο του συστήματος. Ui = Xi * Di , με DI γνωστά απο δεδομένα Αντίστοιχα, μπορούμε να υπολογίσουμε και τον μέσο χρόνο απόκρισης εύκολα ώς:

$$\overline{R_i} = \frac{\text{άθροισμα των χρόνων εξυπηρέτησης}}{\text{αριθμός εξυπηρετήσεων}} = \frac{sum(R)}{len(R)} = \frac{Y_i}{C_i}$$

Όταν συμπληρώνονται 20 κύκλοι, το διάστημα εμπιστοσύνης για το R υπολογίζεται ως εξής: Σύμφωνα με τον παραπάνω τύπο, όταν έχουν συμπληρωθεί K αναγεννητικοί κύκλοι, μία εκτίμηση για το R είναι η:

$$\overline{R} = \frac{\overline{Y}}{\overline{C}}, \text{ όπου } \overline{Y} \text{ και } \overline{C} \text{ οι μέσες τιμές των } Y_i \text{ και } C_i$$

Το διάστημα εμπιστοσύνης υπολογίζεται ως:

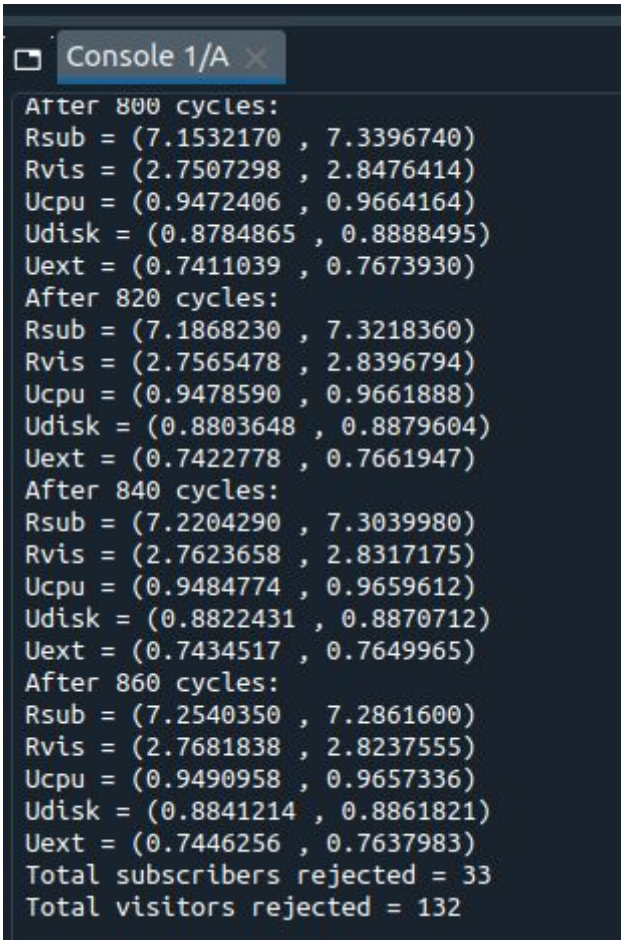
$$P[R - \frac{s \cdot \hat{z}}{\overline{C}\sqrt{K}} \leq r \leq R + \frac{s \cdot \hat{z}}{\overline{C}\sqrt{K}}] = 1 - 0.05$$

Όπου το z είναι το σημείο 1-0.05/2 της κατανομής Student(K-1), και το s υπολογίζεται ως:

$$s^2 = s_y{}^2 + R^2 s_x{}^2 - 2Rs_{yc}$$

Αποτελέσματα προσομοίωσης

Για κάποια εκτέλεση της προσομοίωσης:



Να σημειωθεί πως οι συνολικοί subscribers που δέχτηκε το σύστημα ήταν 549 και από αυτούς οι 33 απορρίφθηκαν. Το ποσοστό απόρριψης της κατηγορίας subscribers είναι 33/549 = 6%. Επίσης, οι συνολικοί visitors που δέχτηκε το σύστημά μας είναι 311 και αυτοί που απορρίφθηκαν από αυτούς ήταν 132. Το ποσοστό απόρριψης της κατηγορίας visitors είναι 42.5%. Αξιοσημείωτο είναι ότι το ποσοστό απόρριψης των visitors είναι αρκετά μεγαλύτερο και αυτό συμβαίνει επειδή οι visitors μπορεί να απορροφηθούν σε οποιοδήποτε στάδιο.

Τα αποτελέσματα συγκεντρωτικά είναι τα εξείς:

Rsub = 7.27
Rvis = 2.79
Ucpu = 0.96
Udisk = 0.88
Uext = 0.75

Παράρτημα: Ο κώδικας της προσομοίωσης

```
#################################################################################################################
# Author            : Stefanos-Stamatis Achlatis,Nikoleta-Markela Iliakopoulou
# Description       : Simulation program
# Requires          : python3, numpy, scipy
# Finalized       : 2019-06-29

#################################################################################################################

import numpy as np
import random
import time
import scipy.stats


#####################################################################
# Random numbers

# to have same results
SEED = 1123412341
np.random.seed(SEED)
random.seed(SEED)
def exp(*args, **kwargs):
    '''Exp(S)'''
    return np.random.exponential(*args, **kwargs)


def erlangk(S, k):
    '''Erlang-k(S) = sum_of_k_independent( Exp(S/k) )'''
    return np.sum(exp(S/k, size=k))

def my_paretovariate():
    '''
    Sample Pareto variate with arbitrary xm
    '''
    U = random.uniform(0,1)
    temp2 = 74.088/(1 - 2*U + U*U)
    final = temp2 ** (1. / 3)
  #  print (f'final is {final}')
    return final

def converge_interval(y, c, a = 0.05):
    '''Given Y, C --> return (R, s*z(1-a/2)/cbar*sqrt(n))'''
    n = len(y)

    ybar = np.average(y)
    cbar = np.average(c)
```

```
    R = ybar / cbar
    S_YY = 1/(n-1) * np.sum((y - ybar) ** 2)
    S_CC = 1/(n-1) * np.sum((c - cbar) ** 2)
    S_YC = 1/(n-1) * np.sum((y - ybar) * (c - cbar))
    s = np.sqrt(S_YY + S_CC*R*R - 2*R*S_YC)

    # find z(1-a/2) point of Student(n-1)
    z = scipy.stats.t.ppf(1-a/2, n-1)

    return (R, s*z/(np.sqrt(n)*cbar))

#####################################################################
# Parameters
VIS_L = 0.72
SUB_L = 1.68
VIS_CPU = 0.021
SUB_CPU = 0.024
VIS_DISC = 0.030
SUB_DISC = 0.025
VIS_EXT = 0.321
SUB_EXT = 0.436
```

```
# Utility


def float_to_str(f):
    '''to string with two decimals'''
    return '%.2f' % f


######################################################################

# State and initialization

class Event:
    '''describes an event'''
    def __init__(self, type, time, category, param=None):
        self.type = type
        self.time = time
        self.category = category
        self.param = param

    def __str__(self):
        time = (float_to_str(self.time))
        return f'Event(time={time}, type={self.type}, category={self.category}, param={self.param})'
```

```
    def __repr__(self):
        return self.__str__()


class Cycle:
    '''describes a completed cycle'''
    def __init__(self, response_times_vis, response_times_sub, duration):
        self.duration = duration
        self.response_times_vis = response_times_vis
        self.response_times_sub = response_times_sub

     #   self.throughput = len(self.response_times) / self.duration

    def __str__(self):
        duration = float_to_str(self.duration)
        count = len(self.response_times_vis + self.response_times_sub)
     #   x = float_to_str(self.throughput)
        meanvis = float_to_str(np.average(self.response_times_vis))
        meansub = float_to_str(np.average(self.response_times_sub))
        return f'Cycle(Rvis={meanvis}, Rsub={meansub}, N={count}, duration={duration})'

    def __repr__(self):
        return self.__str__()

class Simulation:
    def __init__(self):
        # start time of current cycle
        self.cycle_start = 0

        # array of response times for current cycle
        self.response_times_vis = []
        self.response_times_sub = []

        # next arrival times for each client
        self.next_arrival_vis = []
        self.next_arrival_sub = []
        self.next_arrival_vis.append(random.expovariate(VIS_L)) #TSEKARE GENERATE
        self.next_arrival_sub.append(random.expovariate(SUB_L)) #TSEKARE GENERATE
     #   print(self.next_arrival_sub)
     #   print(self.next_arrival_vis)
```

```
        #cpu arrival times for each category
        self.disk_arrival_vis = []
        self.disk_arrival_sub = []

        #cpu arrival times for each category
        self.ext_arrival_vis = []
        self.ext_arrival_sub = []
```

```
        #cpu arrival times for each category
        self.cpu_arrival_vis = []
        self.cpu_arrival_sub = []

        # number of active clients
        self.num_clients = 0

        # list of completed cycles (as Cycle objects)
        self.cycles = []

        self.vis_pareto_time_cpu = []

        self.vis_pareto_time_disk = []

        self.vis_pareto_time_ext = []

        #rejected clients
        self.rejected = 0

        # final result
        self.converged = False

        # confidence interval for R
        self.Rvis = (-np.inf, np.inf)
        self.Rsub = (-np.inf, np.inf)

        # confidence interval for X
        self.X = (-np.inf, np.inf)

    def get_next_event(self):
        '''return next event'''

     #   arsub = self.next_arrival_sub[0]
       # print(arsub)
     #   arvis = self.next_arrival_vis[0]
        #print(arvis)

        if len(self.next_arrival_sub) == 0:
            j = np.inf
        else:
            j = self.next_arrival_sub[0]
        if len(self.next_arrival_vis) == 0:
            i = np.inf
        else:
            i = self.next_arrival_vis[0]
        min_arrival = min(j, i)
        if len(self.cpu_arrival_sub) == 0:
            a = np.inf
        else:
            a = self.cpu_arrival_sub[0]
```

```
        else:
            b = self.cpu_arrival_vis[0]
        if len(self.disk_arrival_sub) == 0:
            c = np.inf
        else:
            c = self.disk_arrival_sub[0]
        if len(self.disk_arrival_vis) == 0:
            d = np.inf
        else:
            d = self.disk_arrival_vis[0]
        if len(self.ext_arrival_sub) == 0:
            e = np.inf
        else:
            e = self.ext_arrival_sub[0]
        if len(self.ext_arrival_vis) == 0:
            f = np.inf
        else:
            f = self.ext_arrival_vis[0]

        next_event = min(min_arrival, a, b, c, d, e, f)
    #  print(f'next_event is {next_event} and is ')
        if next_event == a:
            # assert self.server_up == True
    #       print("a")
            return Event(type='cpu', time=a, category='sub', param=0)#self.cpu_arrival_sub[0])
        elif next_event == b:

     #      print("b")
            return Event(type='cpu', time=b, category='vis', param=0)#self.cpu_arrival_vis[0])
        elif next_event == c:

     #      print("c")
            return Event(type='disk', time=c, category='sub', param=0)#self.disk_arrival_sub[0])
        elif next_event == d:

      #     print("d")
            return Event(type='disk', time=d, category='vis', param=0)#self.disk_arrival_vis[0])
        elif next_event == e:

       #    print("e")
            return Event(type='ext', time=e, category='sub', param=0)#self.ext_arrival_sub[0])
        elif next_event == f:

        #   print("f")
            return Event(type='ext', time=f, category='vis', param=0)#self.ext_arrival_vis[0])
```

```
    elif next_event == min_arrival: #OK

        if i < j:
        #     print("min_arr_vis")
            return Event(type='arrival', time=min_arrival, category='vis', param=0)
        else:
         #    print("min_arr_sub")
          #   print(self.next_arrival_vis)
           # print(self.next_arrival_sub)
            return Event(type='arrival', time=min_arrival, category='sub', param=0)

    else:
        print('too bad')
```

```
def handle_event(self, e=None):
    '''handle a specific event'''

    if e is None:
        e = self.get_next_event()

    # logging.info(e)
    # assert e.time < np.inf
```

```
    if e.type == 'arrival':




        if e.category == 'sub':
            ksi = np.random.normal(40.5, 6)
            if ksi > self.num_clients:
                self.num_clients += 1
                self.cpu_arrival_sub.append(e.time)
                print (self.next_arrival_sub[e.param])
                del self.next_arrival_sub[e.param]
            else:
                del self.next_arrival_sub[e.param]
                self.rejected +=1
        elif e.category == 'vis':
            partime = my_paretovariate()
            if partime > e.time:
                self.num_clients+=1
                self.cpu_arrival_vis.append(e.time)
                self.vis_pareto_time_cpu.append(partime)
                del self.next_arrival_vis[e.param]
            else:
                self.rejected += 1
                del self.next_arrival_vis[e.param]
        else:
            print("rip")

    elif e.type == 'cpu':
        if e.category == 'sub':

            if random.uniform(0,1) < 17/18:
                self.disk_arrival_sub.append(e.time + erlangk(SUB_CPU, 4))
            else:
                self.ext_arrival_sub.append(e.time + erlangk(SUB_CPU , 4))
            del self.cpu_arrival_sub[e.param]
        elif e.category == 'vis':
            if self.vis_pareto_time_cpu[e.param] > e.time:
                if random.uniform(0,1) < 9/10:
                    self.disk_arrival_vis.append(e.time + erlangk(VIS_CPU , 4))
                    self.vis_pareto_time_disk.append(self.vis_pareto_time_cpu[e.param])
                else:
                    self.ext_arrival_vis.append(e.time + erlangk(VIS_CPU , 4))
                    self.vis_pareto_time_ext.append(self.vis_pareto_time_cpu[e.param])
            else:
                self.rejected += 1
            del self.vis_pareto_time_cpu[e.param]
            del self.cpu_arrival_vis[e.param]
        else:
            print("cpurip")
```

```
elif e.type == 'disk':
   # print("disk")
    if e.category == 'sub':

        self.cpu_arrival_sub.append(e.time + exp(SUB_DISC))
        del self.disk_arrival_sub[e.param]
    elif e.category == 'vis':
        if self.vis_pareto_time_disk[e.param] > e.time:
           self.vis_pareto_time_cpu.append(self.vis_pareto_time_disk[e.param])
           self.cpu_arrival_vis.append(e.time + exp(VIS_DISC))

        else:
            self.rejected += 1
        del self.vis_pareto_time_disk[e.param]
        del self.disk_arrival_vis[e.param]
    else:
        print("diskrip")

elif e.type == 'ext':
   # print(self.rejected)
    if e.category == 'sub':

        self.response_times_sub.append(e.time + exp(SUB_EXT))
        del self.ext_arrival_sub[e.param]
        self.num_clients -= 1
    elif e.category == 'vis':
        if self.vis_pareto_time_ext[e.param] > e.time:
           self.response_times_vis.append(e.time + exp(VIS_EXT))
        else:
            self.rejected += 1
        del self.ext_arrival_vis[e.param]
        del self.vis_pareto_time_ext[e.param]
        self.num_clients -= 1
    else:
        print("extrip")
     print(f'e.time is {e.time}')
    self.next_arrival_vis.append(e.time + random.expovariate(VIS_L)) #TSEKARE GENERATE
    self.next_arrival_sub.append(e.time + random.expovariate(SUB_L)) #TSEKARE GENERATE
     print( self.next_arrival_sub)
    if self.num_clients <= 10000 and (self.response_times_vis or self.response_times_sub):
        cycle = Cycle(duration=e.time - self.cycle_start,
                response_times_vis=self.response_times_vis, response_times_sub=self.response_times_sub)

        logging.info(cycle)

        self.cycles.append(cycle)
        self.response_times_sub = []
        self.response_times_vis = []
```

```
        #cpu arrival times for each category
        self.disk_arrival_vis = []
        self.disk_arrival_sub = []

         self.next_arrival_vis.append(e.time + random.expovariate(VIS_L)) #TSEKARE GENERATE
         self.next_arrival_sub.append(e.time + random.expovariate(SUB_L)) #TSEKARE GENERATE

#cpu arrival times for each category
        self.ext_arrival_vis = []
        self.ext_arrival_sub = []

#cpu arrival times for each category
        self.cpu_arrival_vis = []
        self.cpu_arrival_sub = []

        self.vis_pareto_time = []
    #    print("")
        self.cycle_start = e.time

        if len(self.cycles) % 20 == 0:
            if(self.response_times_vis and self.response_times_sub ):
                self.check_convergence(2)
            elif(self.response_times_sub):
                self.check_convergence(1)
            else:
                self.check_convergence(0)
else:
    print("eventrip")
```

```
    def check_convergence(self, flag):
        '''calculate convergence intervals for R and X'''

        # For response times
        if(flag == 2):
            yvis = np.array([np.sum(c.response_times_vis) for c in self.cycles])
            c = np.array([len(c.response_times_vis) for c in self.cycles])
            Rmeanv, Rivv = converge_interval(yvis, c)
            self.Rvis = (Rmeanv - Rivv, Rmeanv + Rivv)
            print('After', len(yvis), 'cycles')
            print('Rvis =', self.Rvis)
        ysub = np.array([np.sum(c.response_times_sub) for c in self.cycles])
        q1 = np.array([c.duration1 for c in self.cycles])
        q2 = np.array([c.duration2 for c in self.cycles])
        q3 = np.array([c.duration3 for c in self.cycles])

        d = np.array([len(c.response_times_sub) for c in self.cycles])
        y1 = np.array([np.sum(c.response_times_cpu) for c in self.cycles])
        y2 = np.array([np.sum(c.response_times_disk) for c in self.cycles])
        y3 = np.array([np.sum(c.response_times_ext) for c in self.cycles])
        Xmean, Xiv = converge_interval(y1, q1)
        Xcpum , Xcpuiv = converge_interval(y2, q2)
        Xdiskm , Xdiskiv = converge_interval(y3, q3)
        Xextm , Xextiv = converge_interval(y, c)
        Ucpum , Xcpuiv = Xcpum*(VIS_CPU*10 + SUB_CPU*18) , Xcpuiv*(VIS_CPU*10 + SUB_CPU*18)
        Udiskm , Udiskiv = Xdiskm*(VIS_DISK*9 + SUB_DISK*17) , Xdiskiv*(VIS_DISK*9 + SUB_DISK*17)
        Uextm , Uextiv = Xextm*(VIS_EXT + SUB_EXT) , Xextiv*(VIS_EXT + SUB_EXT)
        Rmeans, Rivs = converge_interval(ysub, d)
        self.Rsub = (Rmeans - Rivs, Rmeans + Rivs)
        self.Ucpu = (Ucpum - Ucpuiv, Ucpum + Ucpuiv)
        self.Udisk = (Udiskm - Udiskiv, Udiskm + Udiskiv)
        self.Uext = (Uextm - Uextv, Uextm + Uextiv)

      #  print('After', len(yvis), 'cycles')
      #  print('After', len(ysub), 'cycles')
       # print('Rvis =', self.Rvis)
        print('Rsub =', self.Rsub)
        print('Rvis =', self.Rvis)
        print('Ucpu =', self.Ucpu)
        print('Udisk =', self.Udisk)
        print('Uext =', self.Uext)



        # Check convergence
        if self.converged = (2*Rivs < (Rmeans/10)) or len(self.cycles) > 1000:
            print (f'Total subscribers rejected = {self.rejectedsub}')
            print (f'Total visitors rejected = {self.rejectedvis}')
```

```
        print('Rsub =', self.Rsub)
        print('Rvis =', self.Rvis)
        print('Ucpu =', self.Ucpu)
        print('Udisk =', self.Udisk)
        print('Uext =', self.Uext)



        # Check convergence
        if self.converged = (2*Rivs < (Rmeans/10)) or len(self.cycles) > 1000:
            print (f'Total subscribers rejected = {self.rejectedsub}')
            print (f'Total visitors rejected = {self.rejectedvis}')

    def run(self):
        while not s.converged:
            s.handle_event()



if __name__ == '__main__':
    s = Simulation()
    s.run()
```