Ε.Μ.Π., Σχολή Η.Μ.& Μ.Υ.

Ημερομηνία: 24-05-2021

Συστήματα Αναμονής (Queuing Systems) - Ακαδημαϊκό Έτος 2020-2021

4η Ομάδα Ασκήσεων

1. Ανάλυση και Σχεδιασμός τηλεφωνικού κέντρου
2. Σύστημα εξυπηρέτησης με δύο ανόμοιους εξυπηρετητές

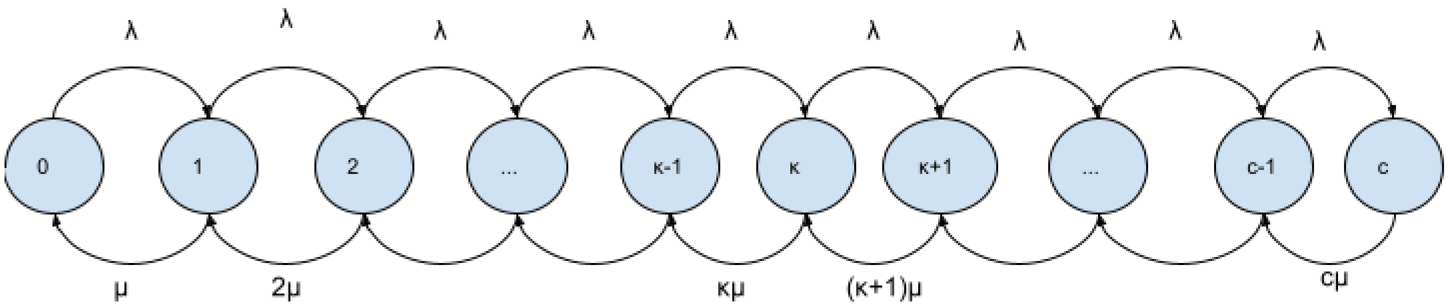
Παραδοτέα: 31-05-2021

Του προπτυχιακού φοιτητή:

Νικήτας Τσίννας, ΑΜ : 03118187

Ανάλυση και Σχεδιασμός τηλεφωνικού κέντρου

1)



Για τον τυχαίο κόμβο κ χρησιμοποιούμε τις εξισώσεις ισορροπίας και επομένως προκύπτει:

$$\lambda \cdot P_k = \kappa\mu \cdot P_{\kappa-1} \Rightarrow P_\kappa = P_{\kappa-1} \cdot (\kappa\mu/\lambda) = (1/\kappa)\rho P_{\kappa-1} \Rightarrow [\, P_\kappa = (\rho^\kappa/\kappa!)*\, P_0 \,] \quad (2)$$

Έπειτα, μέσω της Κανονικοποίησης των Εργοδικών Πιθανοτήτων στην τελευταία σχέση προκύπτει:

$$P_0 + P_1 + \ldots + Pc = 1 \Rightarrow P_0 = 1 / \sum_{\kappa=0}^{c} [\rho^\kappa/\kappa!] \quad (1)$$

Χρησιμοποιώντας τις (1) και (2) προκύπτει η εξίσωση (3) που αντιστοιχεί στην πιθανότητα απόρριψης πελάτη (Pblocking = Pc):

$$P_{blocking} = P_c = (\rho^c/c!) / \sum_{\kappa=0}^{c} [\rho^\kappa/\kappa!] \Rightarrow B(\rho, c) = (\rho^c/c!) / \sum_{\kappa=0}^{c} [\rho^\kappa/\kappa!]$$

Ενώ ο μέσος ρυθμός απωλειών υπολογίζεται από την σχέση: $\lambda - \gamma = \lambda * P_{blocking} = \lambda * (\rho^c/c!) / \sum_{\kappa=0}^{c} [\rho^\kappa/\kappa!]$

Ακολουθεί ο κώδικας υλοποίησης της συνάρτησης erlangb_factorial στο Octave:

```
function Pblocking_factorial = erlangb_factorial (r, c)
  div = 0
  for i=0:c
    div = (div + (r^i/factorial(i)))
  endfor
  Pblocking_factorial = ((r^c / factorial(c))/div)
endfunction
```

Τρέχοντας τον παρακάτω κώδικα, προκύπτει η ακόλουθη έξοδος:

```
clear all
pkg load queueing

B_factorial = erlangb_factorial(10,10)

B_pkg_queueing = erlangb(10,10)
```

```
B_factorial = 0.2146
B_pkg_queueing = 0.2146
```

Επομένως επιβεβαιώνεται η ορθότητα του κώδικα της συνάρτησης.

2)

Για c=0 ισχύει πως $B(\rho, 0) = (\rho^{0}/0!)/\sum_{\kappa=0}^{0}[\rho^{\kappa}/\kappa!] = 1$

Επίσης μπορεί η Β(ρ,c) να γραφτεί και έτσι:

$B(\rho, c) = (\rho^c/c!) / \sum_{\kappa=0}^{c} [\rho^\kappa/\kappa!] \Leftrightarrow B(\rho, c) = 1/ \sum_{\kappa=0}^{c} [\rho^\kappa c! /\kappa!\, \rho^c]$ ή $B(\rho, c) = 1/ \sum_{\kappa=0}^{c} [c! /\kappa!\, \rho^{c-k}]$

Τέλος για c=c+1 έχουμε: $B(\rho, c + 1) = 1/ \sum_{\kappa=0}^{c+1} [(c + 1)! /\kappa!\, \rho^{c+1-k}] = 1/\{ \sum_{\kappa=0}^{c} [(c + 1)! /\kappa!\, \rho^{c+1-k}] + 1\} \Rightarrow$

$\Rightarrow 1/\{ [(c + 1)/\rho] \sum_{\kappa=0}^{c} [c! /\kappa!\, \rho^{c-k}] + 1 \} = 1 / \{ [(c + 1)/\rho] * [1/B(\rho, c)] + 1 \} \Rightarrow \rho B(\rho, c) / [(c + 1) + \rho B(\rho, c)] \Rightarrow$

$\Rightarrow B(\rho, c + 1) = \rho B(\rho, c) / [\rho B(\rho, c) + c + 1]$

Ακολουθεί ο κώδικας της συνάρτησης erlangb_iterative

```
function Pblocking = erlangb_iterative (r, n)
  B = 1
  for i=0:n
    B = (r*B)/((r*B)+i)
  endfor
  Pblocking = B
endfunction
```

Τρέχοντας τον παρακάτω κώδικα προκύπτει η ακόλουθη έξοδος:

```
clear all
pkg load queueing

B_iterative = erlangb_iterative(10,10)

B_pkg_queueing = erlangb(10,10)
```

```
B_iterative = 0.2146
B_pkg_queueing = 0.2146
```

Επομένως επιβεβαιώνεται η ορθότητα του κώδικα της συνάρτησης.

3)
Τρέχοντας τις παρακάτω γραμμές κώδικα προκύπτει η ακόλουθη έξοδος:

```
B_factorial = erlangb_factorial(1024,1024)
B_iterative = erlangb_iterative(1024,1024)
```

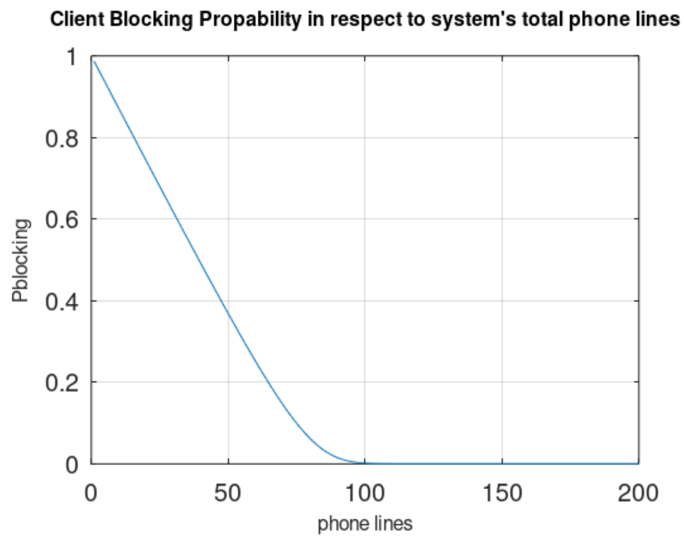
```
>> quesys_ex4_1_comparison

B_factorial = NaN
B_iterative = 0.024524
```

Παρατηρούμε πως η παραγοντική συνάρτηση (factorial) σηματοδοτεί σφάλμα υπερχείλισης. Αυτό συμβαίνει γιατί μέσα στην υλοποίηση της συνάρτησης σε μία βοηθητική μεταβλητή, εδώ η div, αναθέτονται πολύ μεγάλες τιμές τις οποίες δεν μπορεί να διαχειριστεί το υπολογιστικό μηχάνημα.
Αντίθετα, στην αναδρομική συνάρτηση (iterative) δεν υπολογίζεται σε κάθε κάλεσμα της συνάρτησης ξεχωριστά κάποιο παραγαντικό, αλλά χρησιμοποιείται αναδρομή με βάση το $B(\rho,0)= 1$ που οδηγεί στο σωστό αποτέλεσμα χωρίς κάποιο πρόβλημα.

4)
Για την μοντελοποίηση του συστήματος αρχικά υπολογίζουμε τον ρυθμό άφιξης πελατών καθώς και τον ρυθμό εξυπηρέτησης.
Αν η παράμετρος λ (ρυθμός άφιξης πελατών στο σύστημα) είναι 1 κλήση ανά ώρα, δηλαδή ($\lambda=1$), τότε ο μέσος χρόνος εξυπηρέτησης $1/\mu$ θα είναι 23/60 ώρες, δηλαδή 0.383. Επομένως $\rho = \lambda/\mu = 23/60$ για την κάθε τηλεφωνική γραμμή εργαζομένου.

α) Ωστόσο, εφόσον έχουμε στην διάθεσή μας 200 εργαζόμενους, αυτό σημαίνει πως το συνολικό φορτίο που καλείται να εξυπηρετήσει το τηλεφωνικό κέντρο είναι ρ = 200 * 23/60 = 76,66 Erlangs

β)



γ)

```
Minimum number of phone lines for Pblocking to be less than 1% is:
threshhold_lines = 93
```

Ακολουθεί ο κώδικας που χρησιμοποιήθηκε για την εξαγωγή των παραπάνω αποτελεσμάτων:

```
clc;
clear all;
close all;

r = 200 * (23/60); %service rate
c = 1:1:200;       %phone lines
```

```
Pblocking = zeros(1,200); %initialize Pblocking array

for i = 1:1:200
  Pblocking(i) = erlangb_iterative(r,i);
endfor

figure(1)
plot(c,Pblocking)
grid on
title("Client Blocking Propability in respect to system's total phone lines", "fontsize", 8)
xlabel("phone lines", "fontsize", 8)
ylabel("Pblocking", "fontsize", 8)

threshhold_lines = 0;
for i = 1:1:200
  if (Pblocking(i) <= 0.01)
    threshhold_lines = i;
    break;
  endif
endfor

display("Minimum number of phone lines for Pblocking to be less than 1% is: ")
display(threshhold_lines)
```

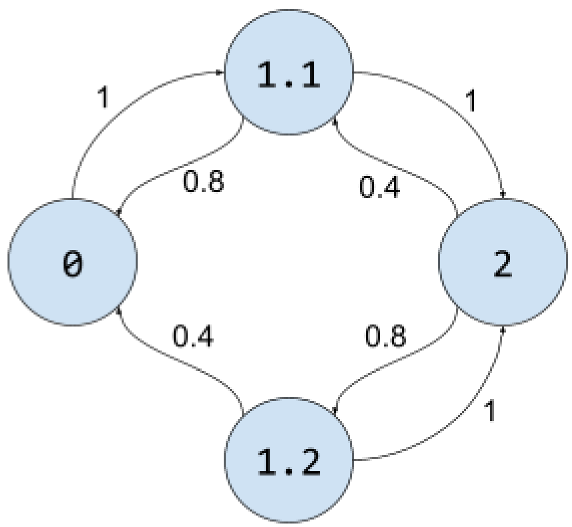
Σύστημα εξυπηρέτησης με δύο ανόμοιους εξυπηρετητές

1)

Το σύστημά αυτό θα έχει 4 καταστάσεις:

- Κατάσταση 0: Καθόλου πελάτες
- Κατάσταση 1.1: Ένας (1) πελάτης στον εξυπηρετητή 1
- Κατάσταση 1.2: Ένας (1) πελάτος στον εξηπυρετητή 2
- Κατάσταση 2: Δύο (2) πελάτες στο σύστημα

Ακολουθεί το διάγραμμα του ρυθμού μεταβάσεων του συστήματος:



- Για τις <u>εργοδικές πιθανότητες</u> ισχύουν οι παρακάτω σχέσεις:
    - $P_0 = 0.8P_{1.1} + 0.4P_{1.2}$
    - $(0.8 + 1)P_{1.1} = P_0 + 0.4P_2$
    - $(1 + 0.4)P_{1.2} = 0.8P_2$
    - Συνθήκη Κανονικοποίησης: $P_0 + P_{1.1} + P_{1.2} + P_2 = 1$

Λύνοντας το παραπάνω σύστημα προκύπτουν οι ζητούμενες τιμές εργοδικών πιθανοτήτων:

$P_0 = 24.95\%$, $P_{1.1} = 21.44\%$, $P_{1.2} = 19.49\%$, $P_2 = 34.11\%$

- Η πιθανότητα απόρριψης πελάτη από το σύστημα ισούται με την πιθανότητα της κατάστασης 2 όπου και οι δύο εξυπηρετητές είναι μη διαθέσιμοι.
  Άρα **Pblocking = 34.11%**

- Ο μέσος αριθμός πελατών στο σύστημα μπορεί να υπολογιστεί ως εξής:
  $E[n] = 1 * (P_{1.1} + P_{1.2}) + 2 * P_2 \simeq (0.2144 + 0.1949) + 2 * 0.3411 = 0.409 + 0.6822 = 1.0912$ πελάτες

2)

- Συμπληρώθηκαν τα κενά του αρχείου demo4.m όπως φαίνεται ακολούθως:

```
threshold_1a = lambda/(lambda+m2);
threshold_1b = lambda/(lambda+m1);
threshold_2_first = lambda/(lambda+m1+m2);
threshold_2_second = (lambda+m1)/(lambda+m1+m2);
```

- Από τον έτοιμο κώδικα που δόθηκε, φαίνεται πως το μόνο κριτήριο σύγκλισης της προσομοίωσης είναι η διαφορά δύο διαδοχικών τιμών μέσου αριθμού πελατών στο σύστημα -ανά 1,000 μεταβάσεις- να είναι μικρότερη του 0.001%.

- Η έξοδος του προγράμματος είναι η ακόλουθη:

```
0.2497
0.2141
0.1953
0.3410
>> |
```

Όπου οι τιμές αυτές αντιστοιχούν στις εργοδικές πιθανότητες αντιστοίχως (P0, P1.1, P1.2, P2). Παρατηρούμε πως οι τιμές της προσομοίωσης πλησιάζουν σε μεγάλο βαθμό τις θεωρητικές τιμές. Η απόκλιση, βέβαια, οφείλεται και στο κριτήριο σύγκλισης. Δηλαδή, αν απαιτήσουμε την διαφορά του μέσου αριθμού πελατών να γίνει 10,000 φορές μικρότερη η προσομοίωση θα διαρκέσει πολύ περισσότερη ώρα και τότε προκύψουν οι ακόλουθες τιμές:

```
0.2494
0.2144
0.1949
0.3413
>>
```

Όπου φαίνεται πως έχουν σχεδόν ταυτιστεί με τις θεωρητικές.

Ακολουθεί ολόκληρος ο κώδικας που χρησιμοποιήθηκε για αυτήν την άσκηση:

```
clc;
clear all;
close all;

lambda = 1;
m1 = 0.8;
m2 = 0.4;

threshold_1a = lambda/(lambda+m1);
threshold_1b = lambda/(lambda+m2);
threshold_2_first = lambda/(lambda+m1+m2);
threshold_2_second = (lambda+m1)/(lambda+m1+m2);

current_state = 0;
arrivals = zeros(1,4);
total_arrivals = 0;
maximum_state_capacity = 2;
previous_mean_clients = 0;
delay_counter = 0;
```

```
time = 0;

while 1
  time = time + 1;

  if mod(time,1000) == 0
    for i=1:1:4
      P(i) = arrivals(i)/total_arrivals;
    endfor

    delay_counter = delay_counter + 1;

    mean_clients = 0*P(1) + 1*P(2) + 1*P(3) + 2*P(4);

    delay_table(delay_counter) = mean_clients;

    if abs(mean_clients - previous_mean_clients) < 0.00001
       break;
    endif
    previous_mean_clients = mean_clients;
  endif

  random_number = rand(1);

  if current_state == 0
      current_state = 1;
      arrivals(1) = arrivals(1) + 1;
      total_arrivals = total_arrivals + 1;
  elseif current_state == 1
    if random_number < threshold_1a
      current_state = 3;
```

```
      arrivals(2) = arrivals(2) + 1;
      total_arrivals = total_arrivals + 1;
    else
      current_state = 0;
    endif
  elseif current_state == 2
    if random_number < threshold_1b
      current_state = 3;
      arrivals(3) = arrivals(3) + 1;
      total_arrivals = total_arrivals + 1;
    else
      current_state = 0;
    endif
  else
      if random_number < threshold_2_first
        arrivals(4) = arrivals(4) + 1;
        total_arrivals = total_arrivals + 1;
      elseif random_number < threshold_2_second
        current_state = 2;
      else
        current_state = 1;
      endif
   endif

endwhile

display(P(1));
display(P(2));
display(P(3));
display(P(4));
```