ΕΠΙΔΟΣΗ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΑΝΑΦΟΡΑ 1ης ΕΡΓΑΣΙΑΣ

ΚΟΛΑΪΤΗΣ ΑΓΓΕΛΟΣ

ΑΜ: 03115029

ΕΙΣΑΓΩΓΗ

Στα πλαίσια της εργασίας έγινε προσομοίωση ενός συστήματος με server-client αρχιτεκτονική, στο οποίο ο server έχει συχνά διαστήματα από downtimes τα οποία αναγκάζουν τους πελάτες σε αναμονή.

Για τις τιμές των διάφορων παραμέτρων που αναφέρονται στην εκφώνηση, μετρήθηκαν ο μέσος χρόνος απόκρισης του συστήματος καθώς και η ρυθμαπόδοσή του.

ΤΕΧΝΙΚΕΣ ΛΕΠΤΟΜΕΡΕΙΕΣ

Το πρόγραμμα αναπτύχθηκε σε γλώσσα Python, με χρήση των βιβλιοθηκών numpy (για διαχείριση πινάκων και υπολογισμών, παραγωγή τυχαίων αριθμών) και 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.

Όσον αφορά τις χωρητικότητες των γραμμών επικοινωνίας μεταξύ server και πελατών, βλέπουμε ότι για N=75 πελάτες η μέγιστη ανάγκη bandwidth είναι 75 * 38Mbps << 8Gbs. Συνεπώς, η γραμμή δεν θα φτάσει ποτέ σε 100% χρησιμοποίηση, άρα δεν υπάρχει κάποια περαιτέρω καθυστέρηση για τους πελάτες.

ΣΥΝΤΟΜΗ ΠΕΡΙΓΡΑΦΗ ΤΗΣ ΥΛΟΠΟΙΗΣΗΣ

Το state του συστήματος που μας ενδιαφέρει για τις ανάγκες τις προσομοίωσης είναι:

- **server_up:** τρέχουσα κατάσταση του server (up/down)
- **num_clients:** αριθμός των πελατών που εξυπηρετούνται από τον server

- **next_downtime:** επόμενος χρόνος στον οποίο ο server θα χαλάσει
- **next_uptime**: επόμενος χρόνος στον οποίο ο server θα επανέλθει

- **next_arrival[N]:** πίνακας του χρόνου της επόμενης αφίξης για κάθε έναν από τους πελάτες. Παίρνει την ειδική τιμή infinite όταν ο πελάτης βρίσκεται ήδη στο σύστημα.
- **next_finish[N]:** πίνακας του χρόνου στον οποίο θα τελειώσει η εξυπηρέτηση κάποιου πελάτη που βρίσκεται στο server. Παίρνει την ειδική τιμή infinite όταν ο πελάτης δεν βρίσκεται στο σύστημα.
- **last_arrival[N]:** πίνακας με τους χρόνους της τελευταίας άφιξης του κάθε πελάτη. Χρησιμεύει ώστε να μπορούμε να υπολογίσουμε τον χρόνο εξυπηρέτησης του, μόλις αυτή ολοκληρωθεί.

- **cycle_start**: χρόνος έναρξης του τρέχοντος αναγεννητικού κύκλου
- **response_times**: λίστα των χρόνων εξυπηρέτησης κατά τον τρέχοντα κύκλο

Ενώ έχουμε τα παρακάτω 4 είδη γεγονότων:

Είδος	Περιγραφή	Ενέργειες
up	Ο server επανέρχεται σε λειτουργία	Υπολογίζονται οι χρόνοι **next_downtime** και **next_uptime**. Αν το σύστημα είναι άδειο, τότε έχει συμπληρωθεί αναγεννητικός κύκλος (βλέπε παρακάτω), οπότε ελέγχουμε και για σύγκλιση. Το **server_up** γίνεται True
down	Ο server χαλάει	Το **server_up** γίνεται False**.** Όλοι οι χρόνοι εξυπηρέτησης των πελατών που βρίσκονται στο σύστημα αυξάνονται κατά το διάστημα που ο server δε θα είναι σε λειτουργία (διότι κατά το διάστημα αυτό δεν εξυπηρετούνται)
arrival	Άφιξη νέου πελάτη (έστω του πελάτη Χ)	Υπολογίζεται το χρονικό διάστημα εξυπηρέτησης του πελάτη και βάσει αυτού ο χρόνος που αυτή θα ολοκληρωθεί (**next_finish[X]**). Φροντίζουμε έτσι ώστε αν ο server δεν είναι σε λειτουργία, ο χρόνος εξυπηρέτησης να αρχίσει να μετράει από την στιγμή που θα επανέλθει. Ο χρόνος άφιξης αποθηκεύεται στον πίνακα **last_arrival[X]**.
finish	Εξυπηρέτηση πελάτη (έστω του πελάτη Χ)	Βάσει του χρόνου άφιξης που έχουμε φυλάξει στο **last_arrival[X]** υπολογίζουμε τον χρόνο απόκρισης που είχε το σύστημα για τον συγκεκριμένο πελάτη. Υπολογίζουμε επίσης τον χρόνο επόμενης άφιξης του (**next_arrival[X]**).

Κατά την αρχικοποίηση, υπολογίζονται οι χρόνοι της πρώτης άφιξης για τους Ν πελάτες του συστήματος και οι χρόνοι της πρώτης βλάβης και διόρθωσης του server. Έπειτα, η κατάσταση ενημερώνεται σύμφωνα με τον προηγούμενο πίνακα.

Για την επιλογή του επόμενου γεγονότος, έχουμε τις χρονικές στιγμές **next_downtime**, **next_uptime**, **next_arrival[N]** και **next_finish[N]**. Επιλέγουμε κάθε φορά την ελάχιστη από αυτές και διαχειριζόμαστε το γεγονός. Αυτή η επιλογή δεν είναι ιδιαίτερα αποδοτική, ωστόσο μας επιτρέπει να διαχειριζόμαστε πολύ εύκολα την καθυστέρηση των εξυπηρετήσεων λόγω των γεγονότων **down**, ενώ οι τιμές των παραμέτρων είναι αρκετά μικρές ώστε η προσομοίωση να τερματίζει σε ελάχιστα δευτερόλεπτα. Σε περίπτωση που είχαμε μεγαλύτερο αριθμό πελατών, ενδεχομένως να συνέφερε η υλοποίηση κάποιας δομής λίστας ή ακόμα και κάποιο Priority Queue.

ΑΝΑΓΕΝΝΗΤΙΚΟΙ ΚΥΚΛΟΙ ΚΑΙ ΕΛΕΓΧΟΣ ΣΥΓΚΛΙΣΗΣ

Όσον αφορά τους αναγεννητικούς κύκλους, θεωρούμε ότι νέος κύκλος ξεκινάει κάθε φορά που ο server επανέρχεται σε κατάσταση UP ενώ είναι άδειος.

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:

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

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:

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

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

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

Το όριο 10% που αναφέρει η εκφώνηση για το μήκος του διαστήματος εμπιστοσύνης αποδείχθηκε πολύ μικρό (η προσομοίωση ολοκληρωνόταν σε μόλις 20 ή 40 κύκλους, κάτι το οποίο δεν είναι πολύ καλό γιατι χρειαζόμαστε μεγάλο αριθμό κύκλων στους υπολογισμούς μας). Συνεπώς, επιλέχθηκε η σύγκλιση να θεωρούμε ότι συμβαίνει όταν το μήκος του διαστήματος εμπιστοσύνης δεν ξεπερνά το 2% της μέσης τιμής του χρόνου απόκρισης.

ΑΠΟΤΕΛΕΣΜΑΤΑ ΕΚΤΕΛΕΣΗΣ

Για κάποια εκτέλεση της προσομοίωσης:

```
$ ./simulation.py
…
After 180 cycles
X = (1.808096941620474, 1.8147023281956218)
R = (2.3988883488205186, 2.447042773225042)
```

Το αποτέλεσμα που παίρνουμε από την προσομοίωση είναι:

$$\text{Ρυθμαπόδοση Συστήματος} = X = 1.81 \text{ πελάτες}/sec$$
$$\text{Χρόνος Απόκρισης Συστήματος} = R = 2.42\ sec$$

Παρατηρούμε ότι ο χρόνος απόκρισης είναι 2.42 sec, μεγαλύτερος του 1.5 sec που είναι κατά μέσο όρο η ανάγκη εξυπηρέτησης του κάθε πελάτη. Συνεπώς, η προσομοίωση μοιάζει να μοντελοποιεί ορθά το σύστημα μας.

ΜΟΝΤΕΛΟΠΟΙΗΣΗ ΜΕ ΧΡΗΣΗ ΤΟΥ ΕΡΓΑΛΕΙΟΥ JMT

Μια ιδέα θα ήταν να κάνουμε την εξής απλοποίηση για το σύστημά μας:

Ο συνολικός χρόνος απόκρισης του κάθε πελάτη αποτελείται από δύο μέρη, την ανάγκη εξυπηρέτησής του, καθώς και τον χρόνο που περιμένουν όσο ο server έχει downtimes.

Αν θεωρήσουμε την λειτουργία του server σε μεγάλο χρονικό διάστημα και εξυπηρετήσεις πολλών πελατών, ο χρόνος εξυπηρέτησης του πελάτη θα είναι η μέση ανάγκη εξυπηρέτησης (S = 1.5 secs) συν τον μέσο χρόνο καθυστέρησης αν τυχόν πετύχει το σύστημα σε βλάβη (B = 4 secs). Για την απλοποίηση των υπολογισμών, θεωρούμε ότι ο πελάτης πετυχαίνει μη λειτουργικό server με πιθανότητα (B / B+A) = (4/24) = 1/6 = 16.66%

Αυτή η σταθερή καθυστέρηση x με κάποια πιθανότητα p, θα μπορούσε να μοντελοποιηθεί ως σταθερή καθυστέρηση x*p για όλους τους πελάτες, κάτι το οποίο θα μπορούσε να γίνει με το JMT προσθέτοντας απλώς στο σύστημα έναν ακόμη κόμβο καθυστέρησης πριν την ουρά εξυπηρέτησης (είτε σταθερού χρόνου x*p, είτε εκθετικού χρόνου με μέση τιμή x*p).

Με βάση τα παραπάνω, ένας απλουστευτικός υπολογισμός για τον μέσο χρόνο απόκρισης θα ήταν:

$$R = 1.5 + \frac{1}{6} * 4 = 1.5 + 0.66 = 2.16\, sec$$

Είναι φανερό ότι η τιμή αυτή απέχει από την τιμή 2.42 που βρήκαμε από την προσομοίωση, ωστόσο προέκυψε σε πολύ λιγότερο χρόνο και μετά από μια, frankly, τουλάχιστον γενναία απλοποίηση (π.χ. Δε λαμβάνουμε καθόλου υπόψιν μας την επιβάρυνση για πελάτες που έχουν την ατυχία να μπλοκάρουν σε δύο/τρία διαδοχικά downtimes του server).