



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΠΙΔΟΣΗ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Ιούνιος 2024

Ιωάννης Γιαννούκος 031 18918
Αργυρώ Τσίπη 031 19950

Εργασία 1

Θέμα 1. Ι. Χρήση του εργαλείου JMT/JMVA

Ορίζουμε τις δύο κατηγορίες στο tab "Classes" του JMT, επιλέγοντάς τες ως closed, αφού

Classes | Stations | Service Times | Visits | Reference Station | What-if | Comment

Classes characteristics
Number, customized name, type of classes and number of customers (closed class) or arrival rate (open class). Add classes one by one or define total number at once. Higher number means higher priority

Number: 2

New Class

*	Name	Type	Priority	No. of Customers	Arrival Rate (λ)
1	CL1	closed	1	304	
2	CL2	closed	1	240	

διαθέτουν και οι δύο χρόνο σκέψης Z.

Συμπληρώνουμε τον αριθμό πελατών που δίνεται από την εκφώνηση ως $N_1 = 304$, $N_2 = 240$.

Στο tab "Stations" ορίζουμε τους 15 σταθμούς, εκ των οποίων ο Db Server CPU είναι load dependent, ενώ οι Clients και Ri είναι load independent. Οι υπόλοιποι σταθμοί είναι delay

Classes | Stations | Service Times | Visits | Reference Station | What-if | Comment

Stations characteristics
Number, customized name and type of stations. Add stations one by one or define the total number at once. Load Dependent stations necessarily require the use of MVA.

Number: 15

New Station

*	Name	Type
1	Clients	Delay (Infinite Server)
2	LAN1	Load Independent
3	LAN2	Load Independent
4	LAN3	Load Independent
5	R1	Delay (Infinite Server)
6	R2	Delay (Infinite Server)
7	Web Server 1 CPU	Load Independent
8	Web Server 1 Disk	Load Independent
9	Web Server 2 CPU	Load Independent
10	Web Server 2 Disk	Load Independent
11	App Server CPU	Load Independent
12	App Server Disk	Load Independent
13	Db Server CPU	Load Dependent
14	Db Server Disk 1	Load Independent
15	Db Server Disk 2	Load Independent

stations.

Για τα Service Times των σταθμών, συμπληρώνουμε τις τιμές με βάση τους πίνακες της εκφώνησης.

Classes	Stations	Service Times	Visits	Reference Station	What-if	Comment
Service Times Input service times of each station for each class. If the station is "Load Dependent" you can set the service times for each number of customers by double-click on "LD Settings..." button. Press "Service Demands" button to enter service demands instead of service times and visits. MULTICLASS MODELS: when for a station the per-class service times are different, the results are correct ONLY IF its scheduling discipline is assumed Processor Sharing (PS) and not FCFS (See BCMP Theorem).						
*		CL1	CL2			
Clients		25.0000	29.0000			
LAN1		0.0320	0.0000			
LAN2		0.0000	0.0900			
LAN3		0.0480	0.1340			
R1		0.0250	0.0000			
R2		0.0000	0.0160			
Web Server...		0.0590	0.0000			
Web Server...		0.0700	0.0000			
Web Server...		0.0000	0.0280			
Web Server...		0.0000	0.0250			
App Server...		0.0480	0.0580			
App Server...		0.0540	0.0660			
Db Server ...	LD Settings...	LD Settings...				
Db Server ...		0.0670	0.0720			
Db Server ...		0.0880	0.0960			

Ειδικά, για τον σταθμό Db Server CPU που είναι load dependent, πατάμε διπλό κλικ στο LD Settings , και θα συμπληρώσουμε στο πεδίο τη συνάρτηση της μέσης συνολικής απαίτησης

$$\text{εξυπηρέτησης } D(K) = \frac{D(1)}{a(k)}, \text{ όπου}$$

Για $n \leq 64$: $a(k) = 40 + 0.60 \cdot n$.

Ενώ για $n > 64$, $a(k) = 38.80$.

Και το $D(1)$ γνωστό από πίνακα της εκφώνησης. Για κατηγορία A το $D(1) = 0.069$, ενώ για κατηγορία B το $D(1) = 0.106$

Επειδή τα δίκτυα μας είναι κλειστά, ο CL1 και ο CL2 θα πρέπει να ανήκουν στο ίδιο LAN το οποίο επιλέξαμε να είναι το LAN1.

Classes	Stations	Service Times	Visits	Reference Station	What-if	Comment
Reference Station The station is used to compute the system throughput and the system response time for each closed class. Performance metrics of open classes are always computed.						
	Class	Station				
	CL1	LAN1				
	CL2	LAN1				

Τελικά, πατάμε solve και εμφανίζονται τα παρακάτω αποτελέσματα:

Για το throughput:

Throughput			
Throughput			
Throughput of each class for each station. System Thr			
*	Aggregate	CL1	CL2
System	0.0258	0.0144	0.0114
Clients	0.0258	0.0144	0.0114
LAN1	0.0258	0.0144	0.0114
LAN2	0.0258	0.0144	0.0114
LAN3	0.0258	0.0144	0.0114
R1	0.0258	0.0144	0.0114
R2	0.0258	0.0144	0.0114
Web Serv...	0.0258	0.0144	0.0114
Web Serv...	0.0258	0.0144	0.0114
Web Serv...	0.0258	0.0144	0.0114
Web Serv...	0.0258	0.0144	0.0114
App Serv...	0.0258	0.0144	0.0114
App Serv...	0.0258	0.0144	0.0114
Db Server...	0.0258	0.0144	0.0114
Db Server...	0.0258	0.0144	0.0114
Db Server...	0.0258	0.0144	0.0114

Για τον αριθμό πελατών:

Number of Customers			
Number of Customers			
Average number of customers for each class at each st			
*	Aggregate	CL1	CL2
Aggregate	544.0000	304.0000	240.0000
Clients	0.6898	0.3601	0.3297
LAN1	4.61E-04	4.61E-04	0.0000
LAN2	0.0010	0.0000	0.0010
LAN3	0.0022	6.93E-04	0.0015
R1	3.60E-04	3.60E-04	0.0000
R2	1.82E-04	0.0000	1.82E-04
Web Serv...	8.51E-04	8.51E-04	0.0000
Web Serv...	0.0010	0.0010	0.0000
Web Serv...	3.18E-04	0.0000	3.18E-04
Web Serv...	2.84E-04	0.0000	2.84E-04
App Serv...	0.0014	6.92E-04	6.60E-04
App Serv...	0.0015	7.79E-04	7.52E-04
Db Server...	543.2964	303.6328	239.6636
Db Server...	0.0018	9.67E-04	8.20E-04
Db Server...	0.0024	0.0013	0.0011

Για τα residence times:

Throughput	Number of Customers	Residence Times	
Residence Times			
Total time spent by each customer class summed across all servers			
*	Aggregate	CL1	CL2
Aggregate	--	--	--
Clients	26.7645	25.0000	29.0000
LAN1	0.0179	0.0320	0.0000
LAN2	0.0397	0.0000	0.0901
LAN3	0.0861	0.0481	0.1343
R1	0.0140	0.0250	0.0000
R2	0.0071	0.0000	0.0160
Web Serv...	0.0330	0.0591	0.0000
Web Serv...	0.0392	0.0701	0.0000
Web Serv...	0.0124	0.0000	0.0280
Web Serv...	0.0110	0.0000	0.0250
App Serv...	0.0525	0.0481	0.0581
App Serv...	0.0594	0.0541	0.0661
Db Server...	21079.9...	21079.8...	21079.9...
Db Server...	0.0693	0.0671	0.0721
Db Server...	0.0917	0.0882	0.0962

Για το system response time:

Throughput	Number of Customers	Residence Times	System Response Time
System Response Time The global aggregate is the "System Response Time" and is calculated as follows: A: This value of System Response Time includes the Residence Times. B: This value of System Response Time does NOT include the Residence Times. Notice: For open classes the Reference Station always coincides with the first station.			
*	Aggregate	CL1	CL2
Aggregate	--	--	--
A	21107.2...	21105.3...	21109.4...
B	21107.1...	21105.3...	21109.4...

Για τον βαθμό χρησιμοποίησης:

Throughput \ Number of Customers \ Residence Times				
Utilization				
Utilization of a customer class at the selected station				
*	Aggregate	CL1	CL2	
Aggregate	--	--	--	
Clients	0.6898	0.3601	0.3297	
LAN1	4.61E-04	4.61E-04	0.0000	
LAN2	0.0010	0.0000	0.0010	
LAN3	0.0022	6.91E-04	0.0015	
R1	3.60E-04	3.60E-04	0.0000	
R2	1.82E-04	0.0000	1.82E-04	
Web Serv...	8.50E-04	8.50E-04	0.0000	
Web Serv...	0.0010	0.0010	0.0000	
Web Serv...	3.18E-04	0.0000	3.18E-04	
Web Serv...	2.84E-04	0.0000	2.84E-04	
App Serv...	0.0014	6.91E-04	6.59E-04	
App Serv...	0.0015	7.78E-04	7.50E-04	
Db Server...	0.0114	0.0000	0.0114	
Db Server...	0.0018	9.65E-04	8.19E-04	
Db Server...	0.0024	0.0013	0.0011	

Για το system power:

System Power				
Aggregate System Power: Aggregate System Throughput divided by the sum of the Residence Times per class weighted by the relative throughputs).				
Per-class System Power: Throughput divided by the Residence Time of the Reference Station.				
A: This value of System Power is computed using the value of the Reference Station.				
B: This value of System Power is computed using the value of the Reference Station.				
Notice: For open classes the Reference Station always contains one customer.				
*	Aggregate	CL1	CL2	
Aggregate	--	--	--	
A	1.22E-06	6.82E-07	5.39E-07	
B	1.22E-06	6.82E-07	5.39E-07	

Τελικά, μας δίνει μία σύνοψη:

JMVA Model Details

Algorithms			
Name	Tolerance	Iterations	Max Samples
MVA	-	-	-

Classes			
Name	Type	Population	Arrival Rate
CL1	closed	304	
CL2	closed	240	

Stations	
Name	Type
Clients	Delay (Infinite Server)
LAN1	Load Independent
LAN2	Load Independent
LAN3	Load Independent
R1	Delay (Infinite Server)
R2	Delay (Infinite Server)
Web Server 1 CPU	Load Independent
Web Server 1 Disk	Load Independent
Web Server 2 CPU	Load Independent
Web Server 2 Disk	Load Independent
App Server CPU	Load Independent
App Server Disk	Load Independent
Db Server CPU	Load Dependent
Db Server Disk 1	Load Independent
Db Server Disk 2	Load Independent

Service Demands		
	CL1	CL2
Clients	25	29
LAN1	0.032	0
LAN2	0	0.09
LAN3	0.048	0.134
R1	0.025	0
R2	0	0.016
Web		
Server 1		
CPU	0.059	0
Web		
Server 1		
Disk	0.07	0
Web		
Server 2		
CPU	0	0.028
Web		
Server 2		
Disk	0	0.025
App		
Server		
CPU	0.048	0.058
App		
Server		
Disk	0.054	0.066
	1.0	1.0
	1.6	1.6
	2.1999999999999997	2.1999999999999997
	2.8	2.8
	3.4	3.4
	3.9999999999999996	3.9999999999999996
	4.6000000000000005	4.6000000000000005
	5.2	5.2
	5.8	5.8
	6.4	6.4
	7.0	7.0
	7.6	7.6
	8.2	8.2

[illegible]

	Services	
	CL1	CL2
Clients	25	29
LAN1	0.032	0
LAN2	0	0.09
LAN3	0.048	0.134
R1	0.025	0
R2	0	0.016
Web Server 1 CPU	0.059	0
Web Server 1 Disk	0.07	0
Web Server 2 CPU	0	0.028
Web Server 2 Disk	0	0.025
App Server CPU	0.048	0.058
App Server Disk	0.054	0.066
Db Server CPU		
Db Server Disk 1	0.067	0.072
Db Server	0.088	0.096

	Visits	
	CL1	CL2
Clients	1	1
LAN1	1	1
LAN2	1	1
LAN3	1	1
R1	1	1
R2	1	1
Web Server 1 CPU	1	1
Web Server 1 Disk	1	1
Web Server 2 CPU	1	1
Web Server 2 Disk	1	1
App Server CPU	1	1
App Server Disk	1	1
Db Server CPU		
Db Server Disk 1	1	1
Db Server Disk 2	1	1

Θέμα 1 ΙΙ. Προγραμματισμός αναλυτικών μοντέλων

Παραθέτουμε τον κώδικα σε C++:

```
#include <iostream>
#include <algorithm> // min()
#include <cmath>     // abs()

using namespace std;

#define DumVal 0
// The 1st element of all the arrays is a dummy value
// so the iteration starts at 1 and ends at its size.

enum TypeOfStation
{
    DELAY,
    LD,
    LI
};

const int M = 15; // αριθμός σταθμών (i)
const int C = 2;  // αριθμός κατηγοριών (j)
const int N1 = 304, N2 = 240;
const int N = N1 + N2; // αριθμός εργασιών (k)
const int n[C + 1] = {DumVal, N1, N2};

double Q[M + 1][C + 1]; // see Bard-Schweitzer approximation (algorithm 5.2)

double Q_old[M + 1][C + 1]; // used for calculating the diversion

double X[C + 1];

const double D[M + 1][C + 1] = {
    {DumVal, DumVal, DumVal},
    /* 1 */ {DumVal, 25.0, 29.0},
    /* 2 */ {DumVal, 0.032, 0.0},
    /* 3 */ {DumVal, 0.0, 0.090},
    /* 4 */ {DumVal, 0.048, 0.134},
    /* 5 */ {DumVal, 0.025, 0.0},
    /* 6 */ {DumVal, 0.0, 0.016},
    /* 7 */ {DumVal, 0.059, 0.0},
    /* 8 */ {DumVal, 0.070, 0.0},
    /* 9 */ {DumVal, 0.0, 0.028},
    /* 10 */ {DumVal, 0.0, 0.025},
    /* 11 */ {DumVal, 0.048, 0.058},
    /* 12 */ {DumVal, 0.054, 0.066},
    /* 13 */ {DumVal, 0.069, 0.106}, // Dij(1) (never explicitly used) except as "D1"
    later
    /* 14 */ {DumVal, 0.067, 0.072},
    /* 15 */ {DumVal, 0.088, 0.096}};
double D_13[C + 1][N + 1];

const TypeOfStation type_of_station[M + 1] = {/* DumVal */ DELAY,
    /* 1 */ DELAY,
    /* 2 */ LI,
    /* 3 */ LI,
    /* 4 */ LI,
    /* 5 */ DELAY,
    /* 6 */ DELAY,
    /* 7 */ LI,
    /* 8 */ LI,
    /* 9 */ LI,
    /* 10 */ LI,
    /* 11 */ LI,
    /* 12 */ LI,
    /* 13 */ LD,
    /* 14 */ LI,
    /* 15 */ LI};

double a[N + 1];
double m[C + 1][N + 1];
double p[N + 1];
```



```

double R[M + 1][C + 1];
double U[M + 1][C + 1];

double maxD(int j)
{
    // return D[13][j];

    double max = D_13[j][1];
    for (int k = 2; k <= n[j]; ++k)
        if (max < D_13[j][k])
            max = D_13[j][k];
    return max;
}

double sumD(int j)
{
    double sum = 0.0;
    for (int i = 1; i <= M; ++i)
        sum += D[i][j];
    return sum;
}

void calc_p13() // depends on Xj, Dij, ak
{
    /* Calculating p13(0|N) */
    double sum = 0.0, product[N + 1];
    for (int k = 1; k <= N; ++k)
    {
        product[k] = 1;
        for (int l = 1; l <= k; ++l)
        {
            double sum_nominator = 0;
            for (int j = 1; j <= C; ++j)
                sum_nominator += (X[j] * D[13][j]);
            product[k] *= sum_nominator / a[l];
        }
        sum += product[k];
    }
    p[0] = 1.0 / (1.0 + sum);

    /* Calculating p13(k|N) */
    for (int k = 1; k <= N; ++k)
        p[k] = p[0] * product[k];
}

int main()
{
    /* Initialization of D_13jk, ak, mjk */
    D_13[1][1] = D[13][1];
    D_13[2][1] = D[13][2];
    for (int k = 1; k <= N; ++k)
    {
        a[k] = (k <= 64) ? 0.40 + 0.60 * k : 38.80;
        for (int j = 1; j <= C; ++j)
        {
            D_13[j][k] = D_13[j][1] / a[k];
            m[j][k] = 1 / D_13[j][k]; // TODO: check later for validity
        }
    }

    /* Initialization of Qij (and Q_old) */
    for (int i = 1; i <= M; ++i)
        for (int j = 1; j <= C; ++j)
            Q_old[i][1] = Q[i][j] =
                type_of_station[i] != DELAY ? n[j] / M : DumVal;

    /* Initialization of Xj */
    for (int j = 1; j <= C; ++j)
        X[j] = min(1 / maxD(j), n[j] / sumD(j));

    bool is_precision_achieved;
    double precision = 0.001;

```

```

do
{
    is_precision_achieved = true;

    calc_p13();

    // R
    for (int i = 1; i <= M; ++i)
        for (int j = 1; j <= C; ++j)
        {
            double sum = 0.0;
            switch (type_of_station[i])
            {
                case DELAY:
                    R[i][j] = D[i][j];
                    break;
                case LI:
                    R[i][j] = D[i][j] * (1.0 + (n[j] - 1) / n[j] * Q[i][j] + (j == 1 ? Q[i][2] :
Q[i][1]));
                    break;
                case LD:
                    for (int k = 1; k <= N; ++k)
                        sum += k / a[k] * p[k - 1];
                    R[i][j] = D[i][j] * sum;
                    break;
            }
        }

    // X
    for (int j = 1; j <= C; ++j)
    {
        double sum = 0.0;
        for (int i = 1; i <= M; ++i)
            sum += R[i][j];
        X[j] = n[j] / sum;
    }

    // Q
    for (int i = 1; i <= M; ++i)
        for (int j = 1; j <= C; ++j)
            Q[i][j] = X[j] * R[i][j];

    /* Precision check */
    for (int i = 1; i <= M; ++i)
        for (int j = 1; j <= C; ++j)
            if (abs(Q[i][j] - Q_old[i][j]) > precision)
            {
                is_precision_achieved = false;
                break;
            }

} while (is_precision_achieved);

/* Calculation of Uij */
for (int j = 1; j <= C; ++j)
    for (int i = 1; i <= M; ++i)
        U[i][j] = X[j] * D[i][j]; // TODO: if U > 1 then U = 1 ??

/* Display Results */ // X,R,Q,U ανά κατηγορία και συνολικά
printf("Ρυθμός απόδοσης Xj:\n\tX_1: %f\n\tX_2: %f\n", X[1], X[2]);
printf("Συνολικός ρυθμός απόδοσης X: %f\n", X[1] + X[2]);

printf("Χρόνος απόκρισης Rj:\n");
double R_[C + 1] = {DumVal, 0.0, 0.0};
for (int j = 1; j <= C; ++j)
{
    for (int i = 1; i <= M; ++i)
        R_[j] += R[i][j];
    printf("\tR_%d: %f\n", j, R_[j]);
}
printf("Συνολικός χρόνος απόκρισης R: %f\n", R_[1] + R_[2]);

```

```

printf("Αριθμός εργασιών Qij:\n");
for (int j = 1; j <= C; ++j)
    for (int i = 1; i <= M; ++i)
        printf("\tQ_%d_%d: %f\n", i, j, Q[i][j]);

printf("Αριθμός εργασιών Qi (sum):\n");
double Q_sum[M + 1];
for (int i = 1; i <= M; ++i)
{
    Q_sum[i] = Q[i][1] + Q[i][2];
    printf("\tQ_%d: %f\n", i, Q_sum[i]);
}

printf("Βαθμός χρησιμοποίησης Uij:\n");
for (int j = 1; j <= C; ++j)
    for (int i = 1; i <= M; ++i)
        printf("\tU_%d_%d: %f\n", i, j, U[i][j]);

printf("Βαθμός χρησιμοποίησης Ui (sum):\n");
double U_sum[M + 1];
for (int i = 1; i <= M; ++i)
{
    U_sum[i] = 0.0;
    for (int j = 1; j <= C; ++j)
        U_sum[i] += U[i][j];
    printf("\tU_%d: %f\n", i, U_sum[i]);
}
}

```

Παρατηρήσεις και Σχόλια:

Ο κώδικας χρησιμοποιεί προσέγγιση Bard-Schweitzer, οπότε τα αποτελέσματα έχουν μία μικρή διαφορά σε σχέση με αυτά της προσομοίωσης του JMT.

Πριν χαρακτηρίσουμε την επίδοση του συστήματος, γνωρίζουμε ότι:

- Μεγάλες τιμές **throughput X** δείχνουν well-performing system, ενώ μικρές τιμές δείχνουν ότι το σύστημα επεξεργάζεται αργά τις εργασίες.
- Χαμηλές τιμές **response time R** είναι επιθυμητές γιατί υποδεικνύουν ότι το σύστημα διαχειρίζεται τις εργασίες efficiently. Ενώ υψηλές τιμές R υποδεικνύουν καθυστερήσεις στο σύστημα.
- Οι **ουρές αναμονής queue length** εάν είναι μικρά σε τιμή μας δείχνουν ότι δεν υπάρχει μεγάλος χρόνος αναμονής για τις εργασίες που περιμένουν να επεξεργαστούν, ενώ μεγάλες ουρές μπορεί και να δημιουργήσουν **bottleneck** στο σύστημα.
- Ο **βαθμός χρησιμοποίησης utilization U** εάν βρίσκεται κοντά στο 1 σημαίνει ότι ο σταθμός είναι πλήρως χρησιμοποιημένος που να μην είναι efficient αλλά μπορεί να οδηγήσει σε **congestion** εάν αυξηθεί η ζήτηση. Εάν είναι <1 τότε είναι **underutilized** που σημαίνει ότι τα resources δεν χρησιμοποιούνται efficiently.

Συνεπώς, για να πούμε εάν το σύστημά μας έχει High, Moderate ή Low performance θα πρέπει: High Performance: High Throughput, Low Response Time, Short Queues, U close to 1 but not exceeding 1.

Moderate Performance: Average Throughput, Moderate Response Time, Manageable Queues, Balanced Utilization.

Poor Performance: Low Throughput, High Response Time, Long Queues, Overloaded or Underutilized Utilization.

Στη δική μας περίπτωση, το σύστημα είναι moderate καθώς υπάρχει ένα bottleneck.

Προτάσεις για βελτίωση:

- Να μοιράσουμε καλύτερα τον φόρτο εργασιών στους σταθμούς για να αποφύγουμε το overload & underutilization. Θα μπορούσε να γίνει με fine-tuning το allocation των tasks στους σταθμούς.

- Να μειώσουμε τα bottlenecks βρίσκοντας τους σταθμούς με $U > 1$ και μεγάλες ουρές και να αυξήσουμε το processing capacity τους. Αυτό θα μπορούσε να πραγματοποιηθεί με το να προσθέσουμε resources ή να βελτιώσουμε την επίδοσή τους.
- Να αλλάξουμε την προτεραιότητα των εργασιών με κάποια strategies ώστε οι σημαντικές εργασίες να επεξεργάζονται γρηγορότερα, μειώνοντας το response time.
- Να βελτιώσουμε το scheduling των εργασιών με κάποιον αλγόριθμο (load balancing algorithm για να κάνουμε distribute τις εργασίες evenly) ώστε να μειώσουμε την ώρα αναμονής (waiting time) και να βελτιώσουμε το throughput.
- Να αλλάξουμε το hardware στους σταθμούς με στενωπό ώστε να βελτιωθούν οι χρόνοι επεξεργασίας των εργασιών.