Universidade de Aveiro

Departamento de Electrónica, Telecomunicações e Informática

Mestrado Integrado Eng. Computadores e Telemática

47064 - Desempenho e Dimensionamento de Redes

Relatório
Desempenho do bloqueio em serviços de video-streaming

Autores :
Guilherme Cardoso 45726
Rui Oliveira 68779

Docente :
Amaro Sousa

Prática :
P2

Ano letivo 2016/2017
Aveiro, 18 de Maio de 2017

Conteúdo

1 Introdução

Este relatório encontra-se estruturado da mesma forma que o guião, seguindo a mesma numeração dos exercícios propostos.

2 Primeira parte

2.1 Estudo analítico: implementação

Para a resolução deste paroblema começámos por implementar o que se encontra descrito no Apêndice A do guia prático.

O método `blocking_probability` permite calcular a probabilidade de bloqueio no sistema descrito.

```
function p = blocking_probability(N, ro)

a= 1; p= 1;
for n= N:-1:1
    a= a*n/ro;
    p= p+a;
end
p= 1/p;
```

O método `average_connection_load` permite calcular a carga média do servidor para este sistema.

```
function o = average_connection_load(N, ro)

a= N;
numerator= a;
for i= N-1:-1:1
    a= a*i/ro;
    numerator= numerator+a;
end
a= 1;
denominator= a;
for i= N:-1:1
    a= a*i/ro;
    denominator= denominator+a;
end
o= numerator/denominator;
```

Ambos os métodos recebem como argumento duas variveis, a varivel N e a ρ (ro) que corresponde à capacidade de carga oferecida e ao fluxo de pedidos, respetivamente.

O *script* seguinte permite calcular todos os valores da probabilidade de bloqueio e da carga média do servidor para cada variável dependente da tabela 1. O resultado é acumulado numa matriz de duas colunas.

```

M = 2;
C_all = [10 10 10 10 100 100 100 100 1000 1000 1000 1000];
lambda_all = [1.0 1.0 1.5 1.5 25 25 30 30 300 300 350 350];
minutes_all = [90 95 90 95 90 95 90 95 90 95 90 95];
result = [];
%format shorte;
format short;
for i= 1:size(C_all,2)
  ro = (lambda_all(i)/60) * minutes_all(i);
  N = floor(C_all(i)/M);
  result(i,1) = blocking_probability(N, ro) * 100;
  result(i,2) = average_connection_load(N, ro) * M ; % ocupacao media em
    mbps e necessario multiplicar por M
end

result
```

2.2 Estudo analítico: resultados

Os resultados seguintes resultam da execução do *script* anterior.

Tabela 1:

Case	λ **(rqts/h)**	**1/** μ **(minutes)**	**C (Mbps)**	**M (Mbps)**	**Blocking Probability %**	**Average Connection Load (Mbps)**
A	1.0	90	10	2	1.4183	2.9575
B	1.0	95	10	2	1.7122	3.1124
C	1.5	90	10	2	5.2074	4.2657
D	1.5	95	10	2	6.0645	4.4619
E	25	90	100	2	0.8740	74.3445
F	25	95	100	2	1.6673	77.8468
G	30	90	100	2	5.4104	85.1306
H	30	95	100	2	7.8176	87.5733
I	300	90	1000	2	0.1234	898.8890
J	300	95	1000	2	1.0636	939.8955
K	350	90	1000	2	6.8500	978.0748
L	350	95	1000	2	11.0772	985.5607

Para todos os casos estudados (A-L) o *throughput* de cada vídeo é sempre 2Mbps. Em relação aos dados obtidos concluímos o seguinte:

- De uma maneira geral, sempre que se aumenta a capacidade de ligação e variável λ (neste contexto pedidos por hora) do processo de Poisson observa-se um aumento da carga média do servidor, como seria de esperar. Mais filmes a serem vistos ou filmes de maior duração geram naturalmente mais tráfgeo.
- Para a mesma capacidade, quanto maior for a variável λdo processo de Poisson (entenda-se, mais pedidos), maior será a probabilidade de bloqueio e a carga média do servidor.
- Em relação aos valores obtidos para a carga média do servidor observa-se um aumento mais ou menos proporcional entre os seguintes casos: A-E-I , B-F-J, C-G-K e D-H-L.
- Menos intuitivo será possivelmente a relação de aumentar o número de pedidos e aumentar o a capacidade do servidor. Talvez intuitivamente aumentando os dois a probabilidade de bloqueio manter-se-ia proporcional, mas acaba ate por reduzir. Isto deve-se a:
  - Um servidor de baixa capacidade bloqueia com um menor número de clientes.
  - Para um menor número de clientes aumenta a probabilidade de todos fazerem o pedido ao mesmo tempo.
  - Num servidor de grande capacidade a probabilidade de todos os clientes pedirem o filme ao mesmo tempo diminui, fazendo que embora o número de pedidos seja alto, a probabilidade de todos os clientes fazerem um pedido ao mesmo tempo diminui, fazendo que a probabilidade de bloqueio seja baixa.

De toda a analise analítica a menos intuitiva foi a conclusão que um aumento de pedidos linearmente proporcional à capacidade leva a uma redução da probabilidade de bloqueio.

2.3 Análise do simulação: implementação

Para a resolução deste problema foi utilizado o simulador disponível no Apêndice B do guia prático.

Para a resolução das alineas seguintes foi implementado o método `simulator1_wrapper` e `confidence_level`. O primeiro permite abstrair o número de corridas do simulador.

```
%% runs simulator1 according to the tests
function [result_b, result_o] = simulator1_wrapper(M, C_all, lambda_all,
    minutes_all, R, runs)
% runs simulator 1 and saves the results
% N: numer of times to run
% *_all: vector of the table to compute results

result_b = zeros(size(C_all,2), runs);  % vector resultados: o 'blocking
    prob'
result_o = zeros(size(C_all,2), runs);  % vector resultados: b 'average
    occupation'

format short;
for it= 1:runs
    for i= 1:size(C_all,2)
        [b, o] = simulator1(lambda_all(i), minutes_all(i), C_all(i), M ,R );
        result_b(i,it) = b*100;
        result_o(i,it) = o;
    end
end

end
```

Para o calculo valores com um determinado intervalo de confiança, implementamos a função *confidence_level()*. Para tal, consultámos o slide 21 da apresentação teórica "Introduction to Discrete Event Simulation".

```
function [media, termo] = confidence_level(alfa, results, N)
% Returns the interval calculated for the given target alfa and data
%
% alfa: confidence level to compute 0 < level < 1
% results: matrix with the values
%
% Returns [media, termo]

media = mean(results);
termo = norminv(1-alfa/2)*sqrt(var(results)/N);

end
```

2.3.1 Alínea a)

Analogamente ao estudo analítico, pretende-se obter valores simulados para todos os casos da tabela.

```
%% alinea 2.2 a)
function ex2_2a(M, C_all, lambda_all, minutes_all)

nr_runs = 10; % nr de simulacoes
R=10000;

[b, o] = simulator1_wrapper(M, C_all, lambda_all, minutes_all, R, nr_runs);

b_confidence = zeros(size(C_all,2),2);
o_confidence = zeros(size(C_all,2),2);

% print results
for i=1:size(C_all,2)
    [b_confidence(i,1), b_confidence(i,2)] = confidence_level(0.1,b(i,:),N);
    [o_confidence(i,1), o_confidence(i,2)] = confidence_level(0.1,o(i,:),N);
    fprintf('Case %c = %.2e +- %.2e || ',char(i+64), b_confidence(i,1),
   b_confidence(i,2))
    fprintf('%.2e +- %.2e\n',o_confidence(i,1), o_confidence(i,2))
end

end

M = 2;
C_all = [10 10 10 10 100 100 100 100 1000 1000 1000 1000];
lambda_all = [1.0 1.0 1.5 1.5 25 25 30 30 300 300 350 350];
minutes_all = [90 95 90 95 90 95 90 95 90 95 90 95];

ex2_2a(M, C_all, lambda_all, minutes_all);
```

2.3.2 Alínea b)

A função de corrida desta simulação é a mesma do exercicio da alínea anterior, com a diferença do número de corridas de 10 para 1000.

2.3.3 Alínea c)

Para esta alínea consideramos R = 10000 e N = 100. Corremos o simulador 100 vezes. Tal como nas alíneas anteriores consideramos o intervalo de confiança a 90%.

Para esta alínea, foi criada uma replica da função simulator1_wrapper com o nome simulator1_new_wrapper que permite o processamento da variável N.

```
%% runs simulator1 according to the tests
function [result_b, result_o] = simulator1_new_wrapper(M, C_all, lambda_all,
    minutes_all, R, N, runs)
% runs simulator 1 and saves the results
% N: numer of times to run
% *_all: vector of the table to compute results

result_b = zeros(size(C_all,2), runs);  % vector resultados: o 'blocking
    prob'
result_o = zeros(size(C_all,2), runs);  % vector resultados: b 'average
    occupation'

format short;
for it= 1:runs
    for i= 1:size(C_all,2)
        [b, o] = simulator1_new(lambda_all(i), minutes_all(i), C_all(i), M,
    R, N);
        result_b(i,it) = b*100;
        result_o(i,it) = o;
    end
end

end
```

```
%% alinea 2.2 c)
function ex2_2c(M)

caseJ_lambda = 300;
caseJ_minutes = 95;
caseJ_C = 1000;

R = 10000;
N= 1000;

[b, o] = simulator1_new_wrapper(M, caseJ_C, caseJ_lambda, caseJ_minutes, R,
    N, 100);

% print results

[b_avg, b_termo] = confidence_level(0.1,b(1,:),N);
[o_avg, o_termo] = confidence_level(0.1,o(1,:),N);
fprintf('Case J = %.2e +- %.2e || ', b_avg, b_termo)
fprintf('%.2e +- %.2e\n',o_avg, o_termo)

end

M = 2;
ex2_2c(M);
```

2.3.4 Alínea d)

Para esta alínea consideramos R = 100000 e N = 1000. Corremos o simulador 10 vezes. Tal como nas alíneas anteriores consideramos o intervalo de confiança a 90%.

```
%% alinea 2.2 d)
function ex2_2d(M, ~, ~, ~)

caseJ_C = 1000;
caseJ_lambda = 300;
caseJ_minutes = 95;

R = 100000;
N= 1000;

[b, o] = simulator1_new_wrapper(M, caseJ_C, caseJ_lambda, caseJ_minutes, R,
    N, 10);

% print results
[b_avg, b_termo] = confidence_level(0.1,b(1,:),N);
[o_avg, o_termo] = confidence_level(0.1,o(1,:),N);
fprintf('Case J = %.2e +- %.2e || ', b_avg, b_termo)
fprintf('%.2e +- %.2e\n',o_avg, o_termo)

end

M = 2;
ex2_2d(M, ~, ~, ~);
```

2.4 Análise da simulação: resultados e conclusões

2.4.1 Alínea a)

Tabela 1:

Case	λ (rqts/h)	1/ μ (minutes)	C (Mbps)	M (Mbps)	Blocking Probability %	Average Connection Load (Mbps)
A	1.0	90	10	2	1.44e+00 +- 9.44e-02	2.97e+00 +- 1.93e-02
B	1.0	95	10	2	1.78e+00 +- 3.98e-02	3.13e+00 +- 2.74e-02
C	1.5	90	10	2	1.78e+00 +- 3.98e-02	4.26e+00 +- 3.28e-02
D	1.5	95	10	2	5.00e+00 +- 1.33e-01	4.45e+00 +- 2.51e-02
E	25	90	100	2	8.39e-01 +- 1.86e-01	7.40e+01 +- 4.08e-01
F	25	95	100	2	1.70e+00 +- 1.60e-01	7.75e+01 +- 2.90e-01
G	30	90	100	2	5.38e+00 +- 2.56e-01	8.47e+01 +- 3.39e-01
H	30	95	100	2	7.58e+00 +- 3.45e-01	8.70e+01 +- 4.17e-01
I	300	90	1000	2	9.80e-02 +- 5.25e-02	8.62e+02 +- 4.08e+00
J	300	95	1000	2	6.33e-01 +- 2.12e-01	8.94e+02 +- 3.63e+00
K	350	90	1000	2	6.08e+00 +- 5.32e-01	9.37e+02 +- 2.87e+00
L	350	95	1000	2	1.00e+01 +- 5.20e-01	9.45e+02 +- 1.36e+00

Os resultados parecem estar em linha com o esperado segundo os resultados analíticos.

Valor de confiança: 90%

Isto quer dizer que aquele valor simulado está com uma probabilidade de 90% dentro do intervalo calculado.

Também se pode observar que existe uma grande variância dos resultados para os diferentes casos, isto é, os valores parecem não ser expressivos para cada uma das simulações.

2.4.2 Alínea b)

Tabela 1:

Case	λ (rqts/h)	1/ μ (minutes)	C (Mbps)	M (Mbps)	Blocking Probability %	Average Connection Load (Mbps)
A	1.0	90	10	2	1.41e+00 +- 8.13e-03	2.95e+00 +- 2.07e-03
B	1.0	95	10	2	1.71e+00 +- 8.93e-03	3.11e+00 +- 2.23e-03
C	1.5	90	10	2	5.21e+00 +- 1.75e-02	4.26e+00 +- 2.76e-03
D	1.5	95	10	2	6.05e+00 +- 1.79e-02	4.46e+00 +- 2.81e-03
E	25	90	100	2	8.67e-01 +- 1.32e-02	7.41e+01 +- 4.95e-02
F	25	95	100	2	1.64e+00 +- 1.80e-02	7.75e+01 +- 4.66e-02
G	30	90	100	2	5.36e+00 +- 3.35e-02	8.48e+01 +- 4.01e-02
H	30	95	100	2	7.74e+00 +- 3.88e-02	8.72e+01 +- 3.31e-02
I	300	90	1000	2	9.95e-02 +- 7.50e-03	8.58e+02 +- 5.83e-01
J	300	95	1000	2	8.59e-01 +- 2.58e-02	8.97e+02 +- 4.64e-01
K	350	90	1000	2	5.93e+00 +- 5.31e-02	9.37e+02 +- 2.23e-01
L	350	95	1000	2	9.74e+00 +- 5.86e-02	9.45e+02 +- 1.84e-01

Nesta simulação, ao começar a atualizar as estatísticas apenas após a chegada de 1000 pedidos permite que a máquina de estados estabilize antes de começar a realizar estatísticas.

A variância dos resultados diminui, isto é, para cada um dos casos simulados os valores começam a tender para a estimativa analítica, assim como parecem diferenciar-se mais de caso para caso.

Pode concluir-se que é essencial que a máquina de estados do simulador esteja estável antes dos valores dos contadores estatísticos serem expressivos e representem os casos da simulação.

2.4.3 Alínea c)

Para esta alínea consideramos R = 10000 e N = 1000. Corremos o simulador 100 vezes. Os resultados obtidos para o caso J foram os seguintes:

Case	λ (rqts/h)	1/ μ (minutes)	C (Mbps)	M (Mbps)	Blocking Probability %	Average Connection Load (Mbps)
J	300	95	1000	2	9.66e-01 +- 2.92e-02l	9.36e+02 +- 4.68e-01

Tabela 1: Resultados obtidos para o caso J com 100 simulações em que R = 10000

Os resultados analíticos para caso J foram:

- **Probabilidade de Bloqueio (Caso J)**: 1.0636 %
- **Carga média do servidor (Caso J)**: 939.8955 Mbps

Comparando os resultados do caso J com as simulações anteriores podemos concluir que um

maior número de corridas permite ter intervalos de variação menores, isto é, para uma probabilidade de 90% do valor estar contido naquele intervalo, o dito intervalo é menor que em todas as simulações anteriores.
Isto deve-se ao facto do número de simulações ser maior. Ao ter mais resultados para fazer uma média é mais estreito o intervalo no qual os valores estarem num certo intervalo.
A ordem de grandeza do intervalo de confiança é, portanto, menor.

2.4.4 Alínea d)

Para esta alínea consideramos R = 100000 e N = 1000. Corremos o simulador 10 vezes. Os resultados obtidos para o caso J foram os seguintes:

Case	**λ (rqts/h)**	**1/ μ (minutes)**	**C (Mbps)**	**M (Mbps)**	**Blocking Probability %**	**Average Connection Load (Mbps)**
J	300	95	1000	2	1.08e+00 +- 9.55e-03	9.40e+02 +- 1.51e-01

Tabela 2: Resultados obtidos para o caso J com 10 simulações em que R = 100000

Comparando os resultados obtidos da Tabela 1, Tabela 2 e os analíticos concluímos que além do intervalo de confiança ser ainda menor também a ordem de grandeza dos valores calculados foi menor.
O nosso palpite é que para o caso J é necessário um maior número de pedidos antes da máquina de estados do simulador estabilizar e que após ter estabilizado os valores resultantes são muito idênticos, visto que foi necessário um menor número de corridas para o intervalo de confiança ser menor.
De todas as simulações, foi a que mais se aproximou aos resultados analíticos.

3 Segunda parte

3.1 Implementação

Para a resolução deste problema procedemos à alteração do simulador 1. Todas as alterações foram realizadas tendo por base as sugestões apresentadas no Apêndice C do guia prático.
Este novo simulador tem portanto implementado um balanceador de carga e permite distribuir a carga por múltiplos servidores (aos quais denominamos por 'S'), assim como parametrizar os critérios de aceitação ou rejeição de filmes de qualidade normal, ou seja, reserva de banda para filmes HD.
A implementação para o simulador 2 é a seguinte:

```
function [b_s, b_h] = simulator2(lambda, p, invmiu, S, W, Ms, Mh, R, N)
%lambda = movies request rate (in requests/hour)
%p = percentage of requests for high-definition movies (in %)
%invmiu= average duration of movies (in minutes)
%S = number of servers (each server with a capacity of 100 Mbps)
%W = resource reservation for high-definition movies (in Mbps)
%Ms = throughput of movies in standard definition (2 Mbps)
%Mh = throughput of movies in high definition (5 Mbps)
%R = number of movie requests to stop simulation
%N = movie request number to start updating the statistical counters

N_C = 100;                    % node capacity
C = S * N_C;

invlambda_S = 60 / ((1-p)*lambda);     % average time between requests (in
    minutes)
invlambda_H = 60 / (p*lambda);     % average time between requests (in
    minutes)

%Events definition:
ARRIVAL_S = 0;
ARRIVAL_H = 1;
DEPARTURE_S = 2;
DEPARTURE_H = 3;

%State variables initialization:
STATE= zeros(1,S);
STATE_S = 0;

%Statistical counters initialization:
NARRIVALS= 0;
NARRIVALS_S = 0;
NARRIVALS_H = 0;

BLOCKED_H= 0;
BLOCKED_S = 0;
```

```
%Simulation Clock and initial List of Events:
Clock = 0;
EventList= [ARRIVAL_S exprnd(invlambda_S) 0; ARRIVAL_H exprnd(invlambda_H)
    0];
EventList= sortrows(EventList,2);

while NARRIVALS < R + N
    event = EventList(1,1);
    Clock = EventList(1,2);
    node_id = EventList(1,3);

    EventList(1,:) = [];

    % process arrivals
    if event == ARRIVAL_H || event == ARRIVAL_S

        % find most available server
        loadbalancer = find(STATE==min(STATE));
        loadbalancer = loadbalancer(1); % pick just one server

        NARRIVALS = NARRIVALS + 1;

        if event == ARRIVAL_S % arrival standard video

            EventList= [EventList; ARRIVAL_S Clock+exprnd(invlambda_S) 0];
            NARRIVALS_S = NARRIVALS_S + 1;

            % ter banda para Standard e reserva disponivel
            if STATE(loadbalancer) + Ms <= N_C && STATE_S + Ms<= (C - W)

                STATE(loadbalancer) = STATE(loadbalancer) + Ms;
                STATE_S = STATE_S + Ms;
                EventList= [EventList; DEPARTURE_S Clock+exprnd(invmiu)
    loadbalancer];

            else
                BLOCKED_S = BLOCKED_S + 1;
            end

        else % arrival high def video

            EventList= [EventList; ARRIVAL_H Clock+exprnd(invlambda_H) 0];
            NARRIVALS_H = NARRIVALS_H + 1;

            % servidor tem que ter pelo menos banda Mh disponivel
            if STATE(loadbalancer) + Mh <= N_C

                STATE(loadbalancer) = STATE(loadbalancer) + Mh;
```

```
                    EventList= [EventList; DEPARTURE_H Clock+exprnd(invmiu)
    loadbalancer];

               else
                    BLOCKED_H = BLOCKED_H + 1;
               end

          end

     else % departures
          if event == DEPARTURE_S
               STATE(node_id) = STATE(node_id) - Ms;
               STATE_S = STATE_S - Ms;
          else
               STATE(node_id) = STATE(node_id) - Mh;
          end


     end

     EventList= sortrows(EventList,2);

     if NARRIVALS == N
          % reset stats and start counting only afer N arrivals
          BLOCKED_S = 0;
          BLOCKED_H = 0;
          NARRIVALS_S = 0;
          NARRIVALS_H = 0;
     end

end

b_s = BLOCKED_S/NARRIVALS_S;
b_h = BLOCKED_H/NARRIVALS_H;

end
```

3.1.1 Alínea a)

O *script* seguinte permite calcular todos os valores de probabilidade de bloqueio em vídeos standard e HD tendo em conta todas as variáveis dependentes da tabela 2.
A função imprime os valores em linha com a tabela.

```
function ex3_a(lambda, S, W)
p = 0.4;
invmiu = 90;
Ms = 2;
Mh = 5;
```

```
R = 10000;
N = 1000;

runs = 40;
test_count = size(lambda,2);

b_s = zeros(test_count, runs);
b_h = zeros(test_count, runs);

b_s_confidence = zeros(test_count,2);
b_h_confidence = zeros(test_count,2);

for test_nr=1:size(lambda,2)
    for lap=1:runs
        [b_s(test_nr,lap), b_h(test_nr,lap)] = simulator2(lambda(test_nr), p, invmiu, S(test_nr), W(test_nr), Ms, Mh, R, N);
    end
    [b_s_confidence(test_nr,1), b_s_confidence(test_nr,2)] = confidence_level(0.1, b_s(test_nr,:), runs);
    [b_h_confidence(test_nr,1), b_h_confidence(test_nr,2)] = confidence_level(0.1, b_h(test_nr,:), runs);

    fprintf('%.6f +- %.6f || %.6f +- %.6f\n', b_s_confidence(test_nr,1)*100, b_s_confidence(test_nr,2)*100, b_h_confidence(test_nr,1)*100, b_h_confidence(test_nr,2)*100);
end

end

lambda = [13 13 13 50 50 50];
S = [1 1 1 3 3 3];
W = [0 60 80 0 180 240];

ex3_a(lambda, S, W)
```

3.1.2 Alínea b)

Neste caso decidimos parametrizar qual seria a reserva de banda limite para a qual decidimos simular, assim como qual é o valor em que a diferença da probabilidade de bloqueios é menor para os dois casos.

```

function ex3_b()
S = 2;                  % 2 server farms
p = 0.1;                % 10% of requests are HD

subscribers = 8000;
lambda = 1 / (24 * 7);  % 1 request / week. lambda is requests/hour
```

```
lambda = lambda * subscribers;

invmiu = 90;
Ms = 2;
Mh = 5;
R = 10000;
N = 1000;          % one month warm up

b_s = zeros(1,100);
b_h = zeros(1,100);

runs = 5;
W_limit = 100;

b_s_confidence = zeros(W_limit,2);
b_h_confidence = zeros(W_limit,2);

for W=1:W_limit
    for lap=1:runs
        [b_s(W,lap), b_h(W,lap)] = simulator2(lambda, p, invmiu, S, W, Ms, Mh, R, N);
    end
    [b_s_confidence(W,1), b_s_confidence(W,2)] = confidence_level(0.1, b_s(W,:), runs);
    [b_h_confidence(W,1), b_h_confidence(W,2)] = confidence_level(0.1, b_h(W,:), runs);

    fprintf('W:%.0f : %.6f +- %.6f || %.6f +- %.6f\n', W, b_s_confidence(W,1)*100, b_s_confidence(W,2)*100, b_h_confidence(W,1)*100, b_h_confidence(W,2)*100);
end

plot(1:W_limit,b_s_confidence(:,1), 1:W_limit, b_h_confidence(:,1));

dist = zeros(1,W_limit);
for i=1:W_limit
    dist(i)= abs(b_s_confidence(i,1) - b_h_confidence(i,1));
end

find(dist==min(dist))

end

lambda = [13 13 13 50 50 50];
S = [1 1 1 3 3 3];
W = [0 60 80 0 180 240];

ex3_b()
```

3.1.3 Alínea c)

Nesta simulação decidimos calcular a malha que representa a probabilidade de bloqueio de um determinado número de servidores *vs* um determinado valor de reserva de banda (W).
Após as simulações, encontrar qual o valor W (reserva de banda) e S (número de servidores) ótimo para uma probabilidade de bloqueio inferior a 0.1% e inferior a 1% no caso de falha de um servidor.

```
function ex3_c()

p = 0.2;          % 20% HD requests
lambda = 1 / (24*7);
lambda = lambda * 17000; % 17000 subscribers
invmiu = 90;
Ms = 2;
Mh = 5;
R = 10000;
N = 1000;

S_limit = 6;
W_limit = 250;

runs = 40;

b_s_confidence = zeros(W_limit, S_limit);
b_s_confidence_error = zeros(W_limit, S_limit);
b_h_confidence = zeros(W_limit, S_limit);
b_h_confidence_error = zeros(W_limit, S_limit);

for S=1:S_limit
    for W=0:W_limit

        b_s = zeros(1,runs);
        b_h = zeros(1,runs);

        for lap=1:runs
            [b_s(lap), b_h(lap)] = simulator2(lambda, p, invmiu, S, W, Ms,
 Mh, R, N);
        end

        [b_s_confidence(W+1,S), b_s_confidence_error(W+1,S)] =
 confidence_level(0.1, b_s, runs);
        [b_h_confidence(W+1,S), b_h_confidence_error(W+1,S)] =
 confidence_level(0.1, b_h, runs);

        fprintf('W %.0f S %.0f: %.5f || %.5f\n', W, S, b_s_confidence(W+1,S)
 , b_h_confidence(W+1,S))
    end
```

```
end

% compute worse case of the two streams
worse_case = zeros(size(b_s_confidence,1), size(b_s_confidence,2));

for i=1:size(b_s_confidence,1)
    for j=1:size(b_s_confidence,2)
        if b_s_confidence(i,j) >= b_h_confidence(i,j)
            worse_case(i,j) = b_s_confidence(i,j);
        else
            worse_case(i,j) = b_h_confidence(i,j);
        end
    end
end

surf(1:S_limit, 0:W_limit, worse_case)
xlabel('Nr Servers');
ylabel('W reservation');
zlabel('Worst case');
%axis([1 S_limit 0 W_limit 0 0.5])
view(70,27)
grid on

% a = [2,2,3;0,2,5;1 2 3]
% [row,column]=find(a==min(min(a(a>0))))

[W_optimal, S_optimal] = find(worse_case==min(min(worse_case(worse_case
    >=0.001))))
menos_um_server=worse_case(:,1:5);
[W_optimal, S_optimal] = find(menos_um_server==min(min(menos_um_server)))

end


ex3_c()
```

3.2 Análise: resultados e conclusões

3.2.1 Alínea a)

Tabela 2

λ (request/hour)	S (N.Server)	W (Mbps)	Standard Blocking (%)	HD Blocking (%)
13	1	0	0.41 +- 0.03	1.30 +- 0.09
13	1	60	1.08 +- 0.08	1.07 +- 0.08
13	1	80	29.13 +- 0.31	0.32 +- 0.03
50	3	0	0.17 +- 0.032	1.77 +- 0.13
50	3	180	0.72 +- 0.06	1.52 +- 0.11
50	3	240	36.28 +- 0.32	0.09 +- 0.03

Através dos dados obtidos podemos tirar as seguintes conclusões:

- Como seria de esperar, quanto maior a reserva de recursos (W, em Mbps) observa-se um aumento da probabilidade de bloqueio em vídeos standard enquanto que, por outro lado, a probabilidade de bloqueio em vídeos HD diminui.
- Também o que se poderia concluir intuitivamente, verifica-se que quanto maior o numero de servidores e o número de pedidos por hora menor será a probabilidade de bloqueio em vídeos standard e maior será a probabilidade de bloqueio em vídeos HD.
- Haverá, portanto, um valor ótimo entre a reserva de banda e o número de servidores para uma probabilidade de bloqueio mínima, tanto em vídeo standard como em vídeo HD, como se poderá verificar nos exercícios seguintes.

3.2.2 Alínea b)

Usando o simulador, é possível calcular para dois servidores as diferentes probabilidades de bloqueio de vídeo standard e HD consoante o valor de reserva (W).

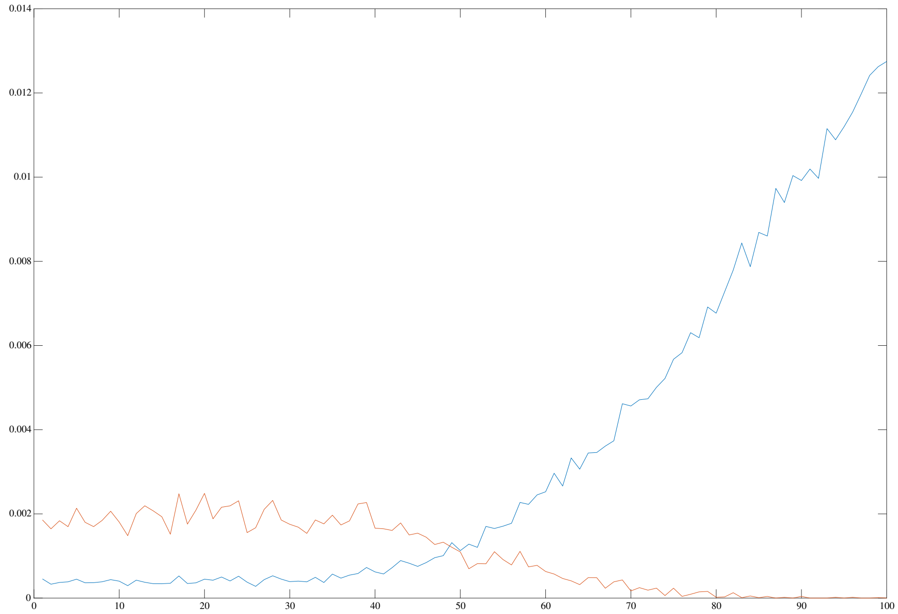

Figura 1: Gráfico comparativo entre a percentagem de bloqueio num vídeo standard e vídeo HD

Legenda:

- **Linha azul**: probabilidade de bloqueio para vídeo standard
- **Linha vermelha**: probabilidade de bloqueio para vídeo HD

Através de observação do gráfico anterior concluímos que a probabilidade de bloqueio para vídeo HD é maior ou igual à probabilidade de bloqueio para vídeo standard quando W é aproximadamente **50**.
Pode-se também ver claramente que os vídeos standard têm uma probabilidade de bloqueio quanto maior for o valor de reserva W, tal como seria de esperar.
Temos então que para W = 50 as probabilidade de bloqueio são as seguintes:

- Probabilidade de bloqueio em vídeo standard: **0.12 +- 0.40**
- Probabilidade de bloqueio em vídeo HD: **0.12 +- 0.43**

Cremos ser este o valor ótimo para minimizar o pior caso de probabilidade de bloqueio. Ficando as probabilidades de tanto para o vídeo standard como para o vídeo HD praticamente as mesmas.

3.2.3 Alínea c) i)

Na nossa implementação é possível observar a malha da pior probabilidade de bloqueio consoante o número de servidores versus a reserva (W).

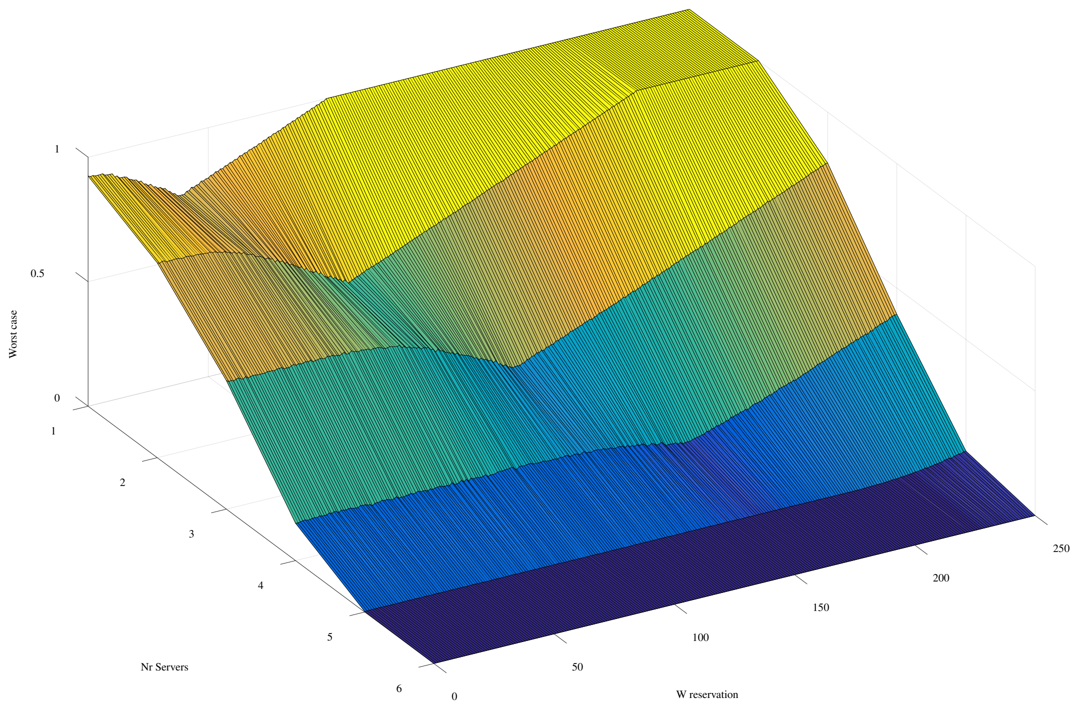

Figura 2: Toda a malha da simulação

Como seria de esperar, observa-se uma descida da probabilidade de bloqueio quanto maior o número de servidores.
Quanto ao valor de reserva W, o pior caso parece estável, pois é estudado o pior caso entre os filmes standard e HD, contudo há claramente um valor ótimo para cada número de servidores disponível. Isto deve-se ao facto em que o valor de bloqueio dos filmes HD tende a diminuir antes do valor de bloqueio dos filmes SD começar a subir.

Para o valor ótimo podemos observar a seguinte zona do gráfico:

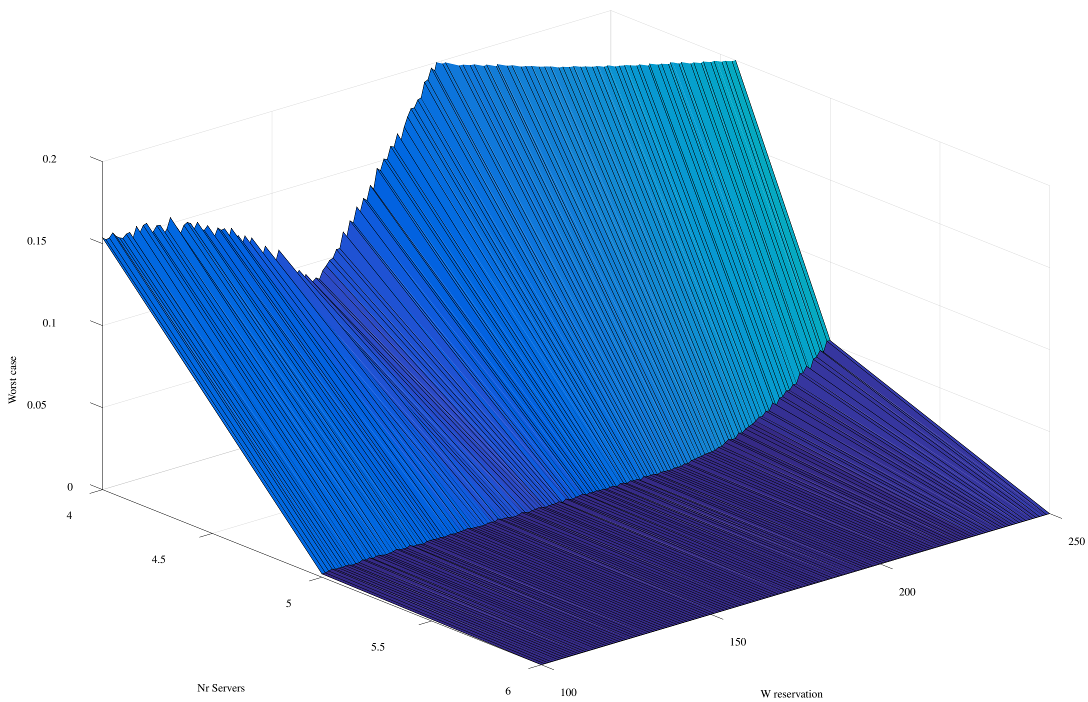

Figura 3: Detalhe do valor ótimo

Podemos concluir que quando número de servidores é 6 o pior caso é praticamente irrelevante. Por outro lado, para 5 servidores há uma ampla zona em que qualquer que seja a variação do W pouco influencia o pior caso. Isto deve-se a um equilíbrio entre o pior caso entre a probabilidade de bloqueio dos filmes standard e dos filmes HD.

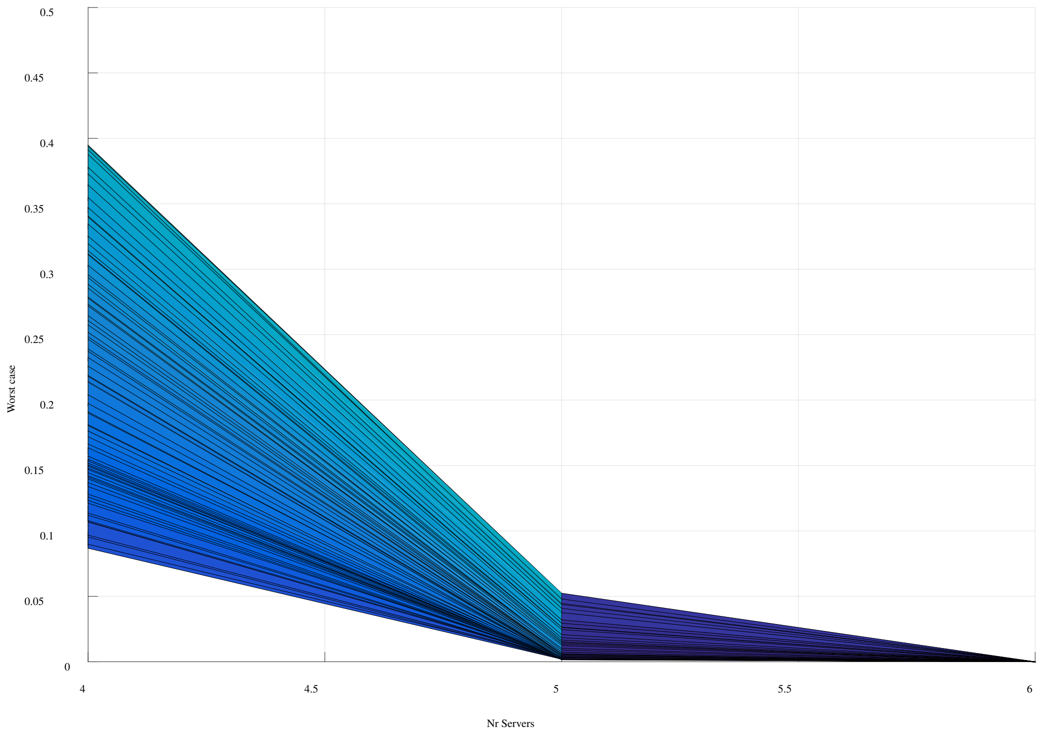

Figura 4: Vista lateral da malha, onde se observa a influência do número de servidores para todos os W na probabilidade de bloqueio.

Contudo, para esta simulação os valores óptimos consoante o pedido (não maior que 0.1% para o pior caso e não maior do que 1% para caso de falha de um servidor).
Assim, deve ser definido como melhor valor de reserva (W) para ser definido no *front-office*: 195 (o valor obtido em várias simulações aproxima-se sempre de 200)
Servidores adicionais de *streaming* de vídeo para campanha de marketing: 5

4 Terceira parte

4.1 Implementação

4.1.1 Alínea a)

Para a resolução deste problema foram seguidos os passos apresentados no apêndice E do guia prático.
Inicialmente procedemos à copia da matriz G (lista de pares de AS ligados) disponível no Apêndice D.
Seguidamente implementámos o algoritmo de labels apresentado por forma a obtermos a matriz I, que dá a distância de cada AS a todos os outros AS.
Por ultimo, procedemos à implementação do algoritmo que nos permitiu gerar o ficheiro ILP. Para tal, tomámos como ponto de partida o exemplo disponível no slide 12 da aula teórica "Optimization based on Integer Linear Programming"e procedemos às devidas adaptações. As restrições aplicadas neste caso foram as seguintes:

$$\text{Minimize} \quad \textstyle\sum_{i=1}^{n} c_i x_i \tag{1}$$

Subject to:

$$\textstyle\sum_{i \in I(j)} y_{ji} = 1 \qquad ,j = 1 \dots n \tag{2}$$

$$y_{ji} \leq x_i \qquad ,j = 1 \dots n, i \in I(j) \tag{3}$$

$$x_i \in \{0,1\} \qquad ,i = 1 \dots n \tag{4}$$

$$y_{ji} \in \{0,1\} \qquad ,j = 1 \dots n, i \in I(j) \tag{5}$$

Figura 5: Restrições para ficheiro ILP.

É importante referir que a restrição (2) terá que aparecer na forma canónica i.e `yij - xi <= 0`

```
G= [ 1 2; 1 3; 1 4; 1 5; 1 6; 1 14; 1 15; 2 3; 2 4; 2 5; 2 7; 2 8; 3 4; 3 5;
    3 8; 3 9; 3 10; 4 5; 4 10; 4 11; 4 12; 4 13; 5 12; 5 13; 5 14; 6 7; 6
    16; 6 17; 6 18; 6 19; 7 19; 7 20; 8 9; 8 21; 8 22; 9 10; 9 22; 9 23; 9
    24; 9 25; 10 11; 10 26; 10 27; 11 27; 11 28; 11 29; 11 30; 12 30; 12 31;
    12 32; 13 14; 13 33; 13 34; 13 35; 14 15; 14 36; 14 37; 14 38; 15 16; 15
    39; 15 40];

% 5 servidores tier 1
n = max(max(G)) - 5;
```

```
N = 40; % nr AS

% custos ignorar tier 1 AS
C(6:15) = 8; % tier 2 AS
C(16:N) = 6; % tier 3 AS

I = zeros(N,N) -1;

v = length(I);

for i=6:size(I,2)
    I(i,i) = 0;
    for a=0:1
        for j=1:size(G,1)
            p1 = I(i,G(j,1));
            p2 = I(i,G(j,2));

            if (p1 == a && p2 == -1)
                I(i,G(j,2)) = a+1;

            elseif (p1 == -1 && p2 == a)
                I(i,G(j,1)) = a+1;
            end

        end
    end

end

%% gerar ILP

fid = fopen('ex3_minimize.lp','wt');
fprintf(fid,'Minimize\n');
for i=6:N
    fprintf(fid,' + %f x%d',C(i),i);
end

fprintf(fid,'\nSubject To\n');
%for j=6:N
for j=6:N
    for i=6:N
        if I(i,j) > -1
            fprintf(fid,' + y%d,%d',j,i);
        end
    end
    fprintf(fid,' =  1\n');
end
for j=6:N
    for i=6:N
```

```
        if I(i,j) > -1
            fprintf(fid,' + y%d,%d - x%d <= 0\n',j,i,i);
        end
    end
end
fprintf(fid,'Binary\n');
for i=6:N
    fprintf(fid,' x%d\n',i);
end
for j=6:N
    for i=6:N
        if I(i,j) > -1
            fprintf(fid,' y%d,%d\n',j,i);
        end
    end
end
fprintf(fid,'End\n');
fclose(fid);
```

Após executarmos o script anterior gerámos o seguinte ficheiro ILP com o seguinte formato:

```
Minimize
 + 8.000000 x6 + 8.000000 x7 + 8.000000 x8 + 8.000000 x9 + 8.000000 x10 +
    8.000000 x11 + 8.000000 x12 + 8.000000 x13 + 8.000000 x14 + 8.000000 x15
    + 6.000000 x16 + 6.000000 x17 + 6.000000 x18 + 6.000000 x19 + 6.000000
    x20 + 6.000000 x21 + 6.000000 x22 + 6.000000 x23 + 6.000000 x24 +
    6.000000 x25 + 6.000000 x26 + 6.000000 x27 + 6.000000 x28 + 6.000000 x29
    + 6.000000 x30 + 6.000000 x31 + 6.000000 x32 + 6.000000 x33 + 6.000000
    x34 + 6.000000 x35 + 6.000000 x36 + 6.000000 x37 + 6.000000 x38 +
    6.000000 x39 + 6.000000 x40
Subject To
 + y6,6 + y6,7 + y6,14 + y6,15 + y6,16 + y6,17 + y6,18 + y6,19 + y6,20 =  1
 + y7,6 + y7,7 + y7,8 + y7,16 + y7,17 + y7,18 + y7,19 + y7,20 =  1
 + y8,7 + y8,8 + y8,9 + y8,10 + y8,21 + y8,22 + y8,23 + y8,24 + y8,25 =  1
 + y9,8 + y9,9 + y9,10 + y9,11 + y9,21 + y9,22 + y9,23 + y9,24 + y9,25 + y9
    ,26 + y9,27 =  1
 + y10,8 + y10,9 + y10,10 + y10,11 + y10,12 + y10,13 + y10,22 + y10,23 + y10
    ,24 + y10,25 + y10,26 + y10,27 + y10,28 + y10,29 + y10,30 =  1
...
 + y6,6 - x6 <= 0
 + y6,7 - x7 <= 0
 + y6,14 - x14 <= 0
 + y6,15 - x15 <= 0
 + y6,16 - x16 <= 0
 + y6,17 - x17 <= 0
 + y6,18 - x18 <= 0
 + y6,19 - x19 <= 0
 + y6,20 - x20 <= 0
 + y7,6 - x6 <= 0
 + y7,7 - x7 <= 0
```

```
...
Binary
 x6
 x7
 x8
 x9
 x10
 x11
 x12
 x13
 x14
 x15
 x16
 x17
...
End
```

4.1.2 Alínea b)

Para a simulação do número de servidores necessário, dividimos os AS por domínios onde estão colocados os *server farm*. Calculamos o pior caso usando os simulador 2.
Resulta, portanto, na seguinte simulação:

```
parte3_bWS()

function parte3_bWS()
p = 0.2;                  % 10% of requests are HD

subscribers = 10*2500 + 24*1000;
lambda = 2 / (24 * 7);  % 1 request / week. lambda is requests/hour
lambda = lambda * subscribers;

invmiu = 90;
Ms = 2;
Mh = 5;
R = 10000;
N = 1000;

%S_limit = 6;
%W_limit = 250;

S_limit = 25;
W_limit = 2500;

runs = 40;

b_s_confidence = zeros(W_limit, S_limit);
b_s_confidence_error = zeros(W_limit, S_limit);
```

```
b_h_confidence = zeros(W_limit, S_limit);
b_h_confidence_error = zeros(W_limit, S_limit);
%23 a 24
for S=23:S_limit
    for W=750:W_limit

        b_s = zeros(1,runs);
        b_h = zeros(1,runs);

        for lap=1:runs
            [b_s(lap), b_h(lap)] = simulator2(lambda, p, invmiu, S, W, Ms, Mh, R, N);
        end

        [b_s_confidence(W+1,S), b_s_confidence_error(W+1,S)] = confidence_level(0.1, b_s, runs);
        [b_h_confidence(W+1,S), b_h_confidence_error(W+1,S)] = confidence_level(0.1, b_h, runs);

        fprintf('W %.0f S %.0f: %.5f || %.5f\n', W, S, b_s_confidence(W+1,S), b_h_confidence(W+1,S))
    end
end

% compute worse case of the two streams
worse_case = zeros(size(b_s_confidence,1), size(b_s_confidence,2));

for i=1:size(b_s_confidence,1)
    for j=1:size(b_s_confidence,2)
        if b_s_confidence(i,j) >= b_h_confidence(i,j)
            worse_case(i,j) = b_s_confidence(i,j);
        else
            worse_case(i,j) = b_h_confidence(i,j);
        end
    end
end

surf(1:S_limit, 0:W_limit, worse_case)
xlabel('Nr Servers');
ylabel('W reservation');
zlabel('Worst case');
%axis([1 S_limit 0 W_limit 0 0.5])
view(70,27)
grid on

% a = [2,2,3;0,2,5;1 2 3]
% [row,column]=find(a==min(min(a(a>0))))

[W_optimal, S_optimal] = find(worse_case==min(min(worse_case(worse_case
```

```
    >=0.001))))
menos_um_server=worse_case(:,1:5);
[W_optimal, S_optimal] = find(menos_um_server==min(min(menos_um_server)))

w=1
end
```

4.1.3 Alínea c)

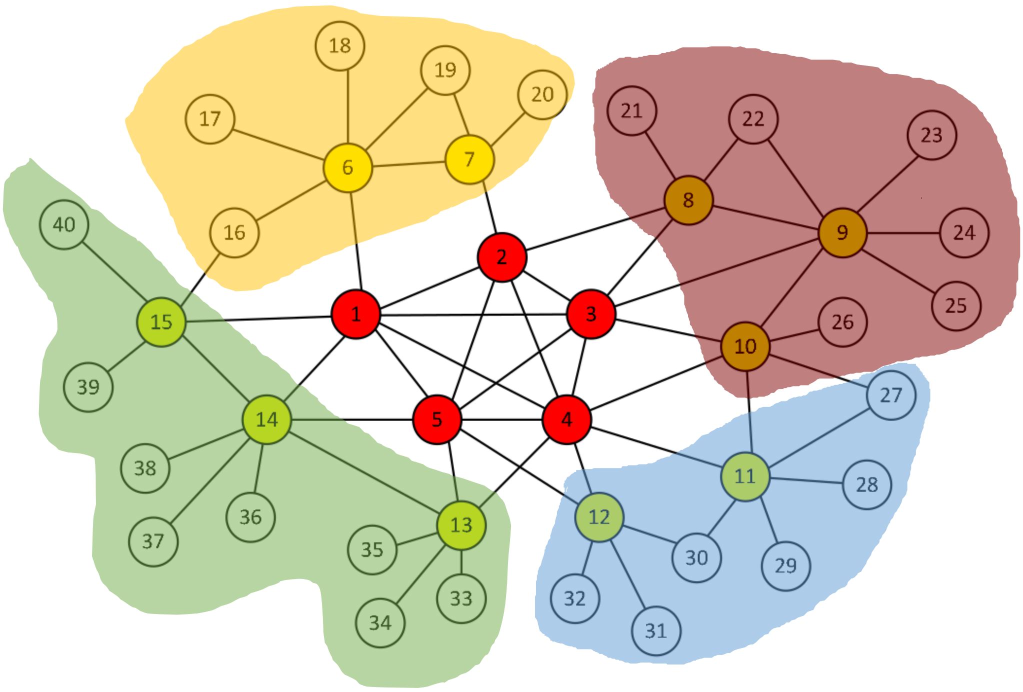

Figura 6: Divisão dos AS por dominios consoante a colocação optima dos server farms

4.2 Análise: resultados e conclusões

4.2.1 Alínea a)

Após submetermos o ficheiro ILP no servidor NEOS obtivemos o seguinte resultado.

```
CPLEX> New value for default parallel thread count: 4
CPLEX> Problem 'cplex.lp' read.
Read time = 0.01 sec. (0.03 ticks)
CPLEX> Tried aggregator 1 time.
Reduced MIP has 292 rows, 292 columns, and 771 nonzeros.
Reduced MIP has 292 binaries, 0 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.00 sec. (0.46 ticks)
Found incumbent of value 230.000000 after 0.00 sec. (0.61 ticks)
Probing time = 0.01 sec. (0.17 ticks)
Tried aggregator 1 time.
Reduced MIP has 292 rows, 292 columns, and 771 nonzeros.
Reduced MIP has 292 binaries, 0 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.00 sec. (0.46 ticks)
Probing time = 0.00 sec. (0.17 ticks)
Clique table members: 292.
MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.
Parallel mode: deterministic, using up to 4 threads.
Root relaxation solution time = 0.07 sec. (0.71 ticks)

Nodes                                              Cuts/
Node  Left     Objective  IInf  Best Integer    Best Bound    ItCnt     Gap

*     0+    0                         230.0000        0.0000
    100.00%
*     0+    0                          60.0000        0.0000
    100.00%
*     0     0      integral     0      28.0000       28.0000      159
    0.00%
Elapsed time = 0.11 sec. (2.54 ticks, tree = 0.00 MB, solutions = 3)

Root node processing (before b&c):
Real time             =    0.11 sec. (2.55 ticks)
Parallel b&c, 4 threads:
Real time             =    0.00 sec. (0.00 ticks)
Sync time (average)   =    0.00 sec.
Wait time (average)   =    0.00 sec.
                          ------------
Total (root+branch&cut) =    0.11 sec. (2.55 ticks)

Solution pool: 3 solutions saved.

MIP - Integer optimal solution:  Objective =  2.8000000000e+01
Solution time =    0.14 sec.  Iterations = 159  Nodes = 0
```

```
Deterministic time = 2.55 ticks (17.89 ticks/sec)

CPLEX> Incumbent solution
Variable Name           Solution Value
x9                            1.000000
x14                           1.000000
x19                           1.000000
x30                           1.000000
y6,14                         1.000000
y7,19                         1.000000
y8,9                          1.000000
y9,9                          1.000000
y10,30                        1.000000
y11,9                         1.000000
y12,14                        1.000000
y13,14                        1.000000
y14,14                        1.000000
y15,14                        1.000000
y16,14                        1.000000
y17,19                        1.000000
y18,19                        1.000000
y19,19                        1.000000
y20,19                        1.000000
y21,9                         1.000000
y22,9                         1.000000
y23,9                         1.000000
y24,9                         1.000000
y25,9                         1.000000
y26,9                         1.000000
y27,9                         1.000000
y28,30                        1.000000
y29,30                        1.000000
y30,30                        1.000000
y31,30                        1.000000
y32,30                        1.000000
y33,14                        1.000000
y34,14                        1.000000
y35,14                        1.000000
y36,14                        1.000000
y37,14                        1.000000
y38,14                        1.000000
y39,14                        1.000000
y40,14                        1.000000
All other variables in the range 1-292 are 0.
```

Através do resultado obtido, concluímos que os ASs que nos dão o caminho mais curto de qualquer Tier-2 ou Tier3 para o server farm mais próximo e que não tenha mais do que um AS intermediário são: **9, 14, 19 e 30**.

Concluímos também que o custo total da solução é **28**. (verificar entrada no ficheiro anterior em

"MIP - Integer optimal solution:")

4.2.2 Alínea b)

Fazendo a simulação análoga aos exercícios anteriores, em que se usa o simulador 2 para encontrar um número ótimo de servidores (S) e de reserva de banda (W), em que cada Tier-2 tem 2500 subscritores e cada Tier-1 tem 1000 subscritores, com 20% de pedidos HD e com 2 filmes por semana por cliente, obtivemos o resultado ideal de **25** servidores e uma reserva de banda de **776**.

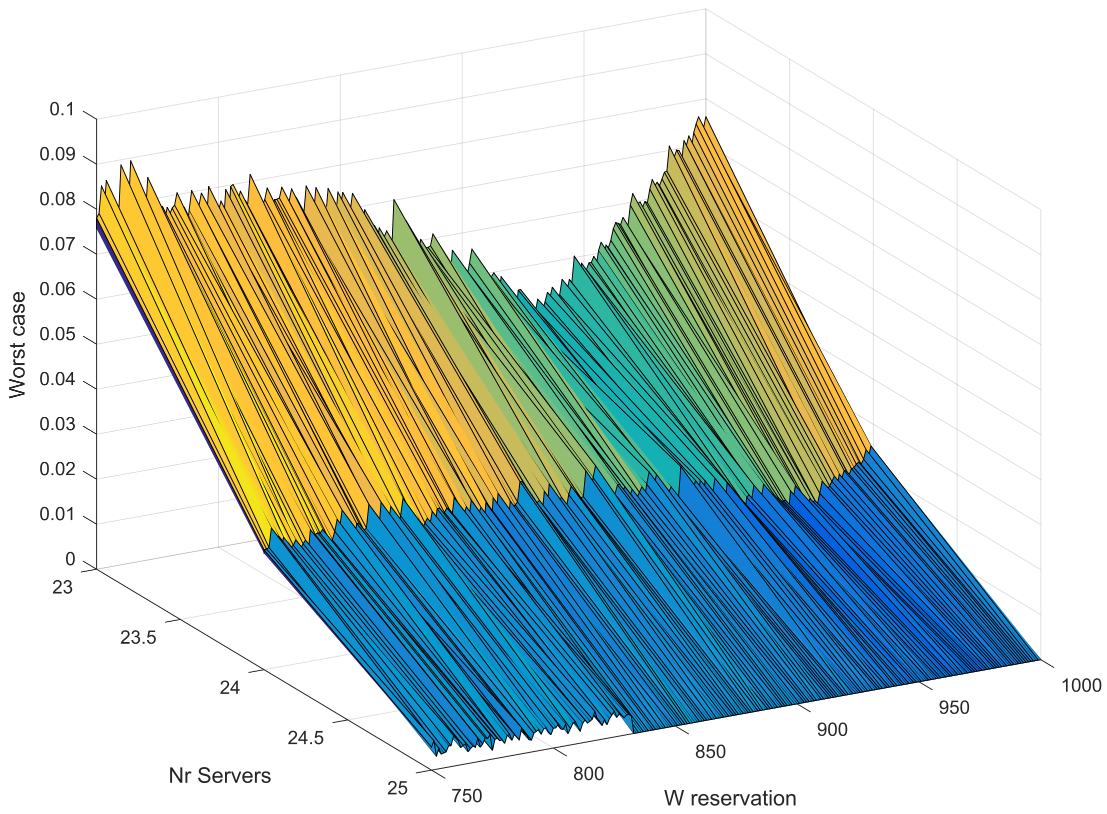

Figura 7: Malha de pior caso número de servidores vs reserva de banda

Contudo, como se pode observar e tal como aconteceu nos exercícios anteriores, a reserva de banda (W) tem uma grande margem de valores para o qual os requisitos são cumpridos.

4.2.3 Alínea c)

Sendo o número de servidor ideais 25 distribuídos por 35 AS, quer dizer que cada AS precisa de cerca de 0.71 servidores.

Segundo a divisão que fizemos:

Server Farm	Número de AS no domínio	Servidores atribuídos
13	8	7
14	11	8
19	7	5
30	9	7

Assumindo que não partilhamos servidores na nosso domínio de divisão, precisamos então de um mínimo de 27 servidores para corresponder à probabilidade máxima de bloqueio de 1%.

5 Referências

- Guia prático disponível na página elearning da disciplina
- Slide teóricos disponível na página elearning da disciplina
- Documentação matlab `https://www.mathworks.com/help/matlab/`