INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Simulação de Redes de Actividades em Java

Aplicação à Estabilização de Redes de Filas de Espera

Marta Fraústo Basso Rebello

Dissertação para obtenção do Grau de Mestre em

Engenharia Electrotécnica e de Computadores

Júri

Presidente: Doutor Francisco Miguel Prazeres Silva Garcia

Orientador: Doutor Carlos Filipe Gomes Bispo

Vogais: Doutor Fernando Henrique Corte Real Mira da Silva

Doutor Pedro Manuel Urbano de Almeida Lima

Dezembro de 2007

Resumo

Esta tese tem como objectivo principal o desenvolvimento de um pacote de simulação de redes de actividades em Java. O simulador foi modelado com base no conceito de rede de actividades, desenvolvido por Harrison e com algum contributo por parte desta tese no sentido de englobar as possibilidades de merge, split e utilização de mais de uma unidade de material, mesmo que este venha de um só buffer. Tratando-se de simulação estocástica discreta foram definidos os eventos que a regem, bem como as suas variáveis aleatórias. O simulador funciona com base num ficheiro de entrada seleccionado a partir de uma interface gráfica e de alguns parâmetros que esta interface requisita.

Foram implementadas várias políticas de sequenciamento para os servidores bem como a opção de permitir que numa rede, servidores distintos tenham políticas distintas.

A simulação produz vários ficheiros de saída de modo a que se torne possível estudar o comportamento da rede perante os parâmetros entrada, como o cálculo de tempos de ciclo restantes, estudo de estabilidade, desempenho de várias políticas, etc. Todo o pós processamento foi efectuado em MatLab, mas fica ao critério do utilizador escolher o método que achar mais conveniente, uma vez que os dados de saída contêm toda a informação relativamente à simulação.

Entre as políticas implementadas está um mecanismo de bloqueio a buffers com o objectivo de mostrar que, em alguns casos, é vantajoso para a rede que um servidor se mantenha inactivo mesmo na presença de clientes, alargando assim, ao conjunto de políticas óptimas, políticas idling.

Palavras Chave

Redes de Actividades, Redes de Filas de Espera, Simulação de Eventos Discretos, Políticas de Sequenciamento, Estabilidade.

Abstract

The main purpose of this thesis is the development of a simulation package of activity networks in Java. The simulator was modeled using the concept of activity networks developed by Harrison, and with some added features which are contributions of this thesis, such as: the possibility to represent activities which have merge, split, and may consume more than one unit of material, even if it comes from the same buffer. Because this is stochastic discrete simulation the events that rule the simulation, as well as its random variables, were defined. To run the simulator, an input file is selected through a graphical interface and some other parameters are required to be defined by the user. Several scheduling policies were implemented and, for any system, each server may use its own scheduling policy. The simulation produces several output files in order to study the network behavior, regarding the input parameters, like the remaining cycle time, stability of the network, performance of the chosen policies, etc. The data collected from the simulation was processed in Matlab, but the user has the possibility to chose another program. For all the scheduling policies that were programmed, it is possible to add a blocking mechanism, with the purpose to show that, in some cases, there are advantages in keeping the server idle in the presence of clients waiting to be processed, thus expanding the set of possible policies to study, including idling policies.

Key Words

Activity Networks, Queuing Networks, Discrete Event Simulation, Scheduling Policies, Stability.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Todos nós no nosso dia-a-dia passamos por sistemas de filas de espera. Para o utilizador, quanto menor for o tempo de espera melhor se considera o serviço. A maior parte destes sistemas já são hoje estudados precisamente para poder, de alguma forma, optimizar estes tempos. Por exemplo, nas lojas do cidadão já existe alguma previsão de quando a pessoa vai ser atendida, os supermercados têm filas para clientes só com um cesto, os correios funcionam com regime de fila única para evitar os 'azares' de ter alguém à nossa frente com um expediente sem fim enquanto que o cliente que chegou depois de nós, e se colocou na fila do lado, já está a ser atendido, etc.

Além deste tipo de serviços existem, em muitos outros contextos, sistemas em que se quer reduzir ao máximo o tempo de espera. Uma fábrica que recebe matéria prima para depois a transformar num produto final é um exemplo que vai ser aqui estudado.

Foi desenvolvido um pacote em Java para fazer a simulação estocástica discreta de redes de actividades. O conceito de rede de actividades foi introduzido por Harrison, [2], e é um conceito mais abrangente do que o de rede de filas de espera. Permite a possibilidade de acções como merge (quando uma actividade utiliza recursos de mais do que um buffer), split (quando o resultado de uma actividade é colocado em mais de um buffer, ou seja, existe mais do que uma unidade no final) e, finalmente, utilização de mais do que uma unidade de material do mesmo ou de mais do que um buffer para efectuar uma actividade.

A razão da criação do simulador foi a de se necessitar de uma ferramenta que fosse flexível para poder testar diferentes políticas de sequenciamento, tentando, se possível, melhorar os tempos de ciclo dos clientes e com isso o seu tempo de permanência no sistema, e, estudar também possíveis políticas para a estabilização de redes instáveis.

O simulador abarca todas de configurações de redes já referidas, e ainda a possibilidade de escolher

a política de cada um dos servidores da rede, número de simulações e critério de paragem pretendido, possibilidade de ter ou não controlador activo e ainda uma série de outros parâmetros relativos à rede e respectivas políticas de sequenciamento que se pretende simular.

Durante a simulação são criados ficheiros de saída com toda a informação relevante para se poder analisar o comportamento da rede simulada. O pós processamento foi realizado em Matlab para não sobrecarregar o simulador com cálculos que nem sempre são relevantes, e deste modo, permitir que as simulações propriamente ditas demorassem o menos tempo possível.

Desenvolveu-se um controlador que obriga a que cada servidor divida o seu tempo pelos diferentes buffers que serve. Impondo um limite de tempo dedicado a cada um dos buffers da rede, de modo que, um buffer, mesmo não tendo prioridade sobre outro, não esteja demasiado tempo sem ser processado. Estes limites podem fazer com que, em alguns casos, um servidor fique inactivo, mesmo na presença de clientes à espera de serem processados.

Este controlador tem como objectivo a estabilização de redes que, sob determinadas políticas, têm um comportamento instável. Mais do que a estabilização, pretende-se melhorar os tempos médios de ciclo da política que é actualmente considerada como sendo a que tem um melhor desempenho a esse nível, ou seja, a política FSVCT.

Para o caso em que o objectivo é a estabilização de redes instáveis utilizou-se uma rede já estudada por Thomas L. Seidman, [3], e posteriormente por José Moreira na sua tese de mestrado, [4].

Para o segundo caso, em que o objectivo é o melhoramento dos tempos médios de ciclo utilizaram-se os mesmos dados que Lu *et al.* em [1].

1.1 Estrutura da Dissertação

Esta tese encontra-se dividida em 6 capítulos. Neste capítulo pretende-se fazer uma introdução ao problema e ao trabalho desenvolvido, bem como um breve resumo dos conteúdos dos restantes capítulos.

O segundo capítulo contém duas secções. Na primeira faz-se uma revisão da literatura dos pontos importantes para a compreensão do trabalho efectuado e na segunda introduz-se o conceito de rede de actividades e a sua respectiva modelação.

O terceiro capítulo apresenta o simulador implementado. A primeira secção fala do seu funcionamento em geral, dos eventos, e das leis que regem a simulação. Nas secções seguintes explica-se de modo detalhado o formato de um ficheiro de entrada, da interface e dos respectivos ficheiros de saída.

No quarto capítulo começa-se por explicar, na primeira secção, a importância de obrigar a que os

servidores multi-classe distribuam o seu tempo por quotas, definidas de acordo com a carga imposta por cada buffer a cada servidor, independentemente das prioridades dos buffers que servem. Na segunda secção discute-se o controlador implementado em detalhe.

O quinto capítulo contém os resultados obtidos nas várias simulações, e a sua respectiva análise. Existem duas secções, uma para cada uma das redes estudadas. A primeira visa a estabilização de uma rede, e a segunda, o melhoramento dos tempos médios de ciclo.

Finalmente, o sexto e último capítulo discute as principais conclusões do trabalho desenvolvido, bem como uma série de ideias para desenvolver num trabalho futuro.

Capítulo 2

Redes de Actividades

Neste capítulo faz-se um pequeno resumo da teoria mais importante para esta tese, [5], e também se introduzem alguns conceitos essenciais para a compreensão dos restantes capítulos. Na secção 2.1 faz-se uma revisão de conceitos sobre redes de filas de espera, com especial atenção naqueles que são mais relevantes nesta tese, mais concretamente, o que é uma rede de filas de espera, políticas de sequenciamento dos servidores, desempenho e estabilidade de redes. Na secção 2.2 caracteriza-se uma rede de actividades bem como as vantagens e limitações deste tipo de caracterização.

2.1 Caracterização de Redes de Filas de Espera

Uma rede de filas de espera caracteriza-se por ter:

- Servidores (um ou mais);
- Buffers (que alimentam os servidores).

Servidor: alimenta-se de um buffer de entrada, e, após terminar o serviço, coloca o cliente num buffer de saída ou então, não coloca em buffer nenhum, se for o caso em que o cliente sai do sistema. Um servidor caracteriza-se por ter uma taxa de atendimento, μ, distinta por cada tipo de cliente que processa. Este valor traduz o número médio de clientes que o servidor processa numa unidade de tempo. Por sua vez o tempo de serviço segue normalmente uma distribuição exponencial com média $1/\mu$, designado daqui para a frente como MPT (mean processing time). Cada servidor, no caso de servir mais do que um buffer distinto precisa de ter uma política de sequenciamento, ou seja, um critério de decisão de qual o buffer a servir em seguida. Essa política pode ser FIFO, Random, etc.

Buffer: um buffer é simplesmente uma fila de clientes, onde estes se acumulam, esperando pela sua vez de serem atendidos. Estes clientes dentro do buffer estão sempre ordenados por tempo de chegada. Dependendo da politica de sequenciamento utilizada pelo servidor, os buffers podem ou não ter prioridade sobre outros buffers. O processo de chegada num sistema de filas de espera é normalmente descrito como um processo de Poisson (embora possa ser outro) e, no contexto desta tese, assume-se estacionário. Cada buffer tem então uma taxa de chegada λ que indica quantos clientes chegam por unidade de tempo, analogamente ao caso do servidor $1/\lambda$ indica o tempo médio entre chegadas ao buffer.

2.1.1 Políticas de Sequenciamento dos Servidores

Cada servidor de uma rede tem de ter definida a sua política de sequenciamento. As políticas têm dois aspectos que as caraterizam:

- Localização. O servidor decide que buffer servir em seguida dependendo:
    1. Local: apenas de informação local ou dos seus clientes;
    2. Não local: de informação contida na rede.
- Idleness:
    1. Idling: cada servidor tem a capacidade de ficar inactivo mesmo na presença de clientes à espera de serem processados;
    2. Non-idling: um servidor não pára enquanto tiver clientes à espera de serem processados.

Nesta tese é estudado o desempenho de políticas locais e não locais em várias redes. As políticas de decisão local usadas foram: FIFO (first in first out) e LBFS (last buffer first served) e as de decisão não local foram políticas de least slack, mais concretamente uma subclasse destas políticas chamada Fluctuation Smoothing Policies. Estas políticas visam reduzir a variância do tempo de ciclo denominada FSVCT e o tempo médio de ciclo denominada FSMCT e são explicadas em detalhe por Lu *et al*, [1]. Em rigor, estas últimas políticas, apesar de requererem informação do sistema para funcionarem, a informação em causa é distribuível, na medida em que cada cliente a pode transportar consigo. Outras políticas existem em que a informação contida na rede, não sendo distribuível por ser dinâmica, implica que uma tomada de decisão por parte de um servidor requer sempre troca de informação com outros servidores ou com um decisor central.

É importante realçar que a política FIFO foi aqui referida como sendo local, mas existem duas implementações possíveis, uma absoluta e outra relativa. No caso de ser relativa significa que o servidor

escolhe processar o buffer cujo primeiro cliente está à espera daquele servidor em concreto à mais tempo. Se a política for absoluta significa que o servidor em questão escolhe processar o buffer cujo primeiro cliente está à mais tempo no sistema. Resumindo, para a política relativa o que conta é o tempo de chegada ao buffer, e, para a política absoluta o que conta é o tempo de chegada ao sistema.

2.1.2 Desempenho e Estabilidade de redes

É extremamente importante medir o desempenho destas redes. A qualidade de serviço dos sistemas depende do tempo de serviço e do tempo de espera. Normalmente o que se pode optimizar é o tempo de espera, uma vez que o melhoramento do tempo de serviço pode ser muito difícil de reduzir (por exemplo no caso de consultas médicas) ou então corresponde a um investimento significativo (no caso de uma fábrica por exemplo, máquinas que consigam fazer o mesmo em menos tempo).

A carga imposta a cada servidor, ou intensidade de tráfego, é calculada pela fracção entre a taxa de chegada a um buffer e a respectiva taxa de processamento. Na equação seguinte, λ_i^k e μ_i^k correspondem à taxa de chegada e à taxa de processamento do servidor i para o buffer k.

$$\rho_i = \sum_{k=1}^{N} \frac{\lambda_i^k}{\mu_i^k} \times c(i,k) < 1, \tag{2.1}$$

em que N é o número de buffers existentes na rede e em que $c(i,k)$ é o representado na Equação 2.2.

$$c(i,k) = \begin{cases} 1 & \text{se } i \text{ processa } k \\ 0 & \text{c.c.} \end{cases} \tag{2.2}$$

Também é essencial garantir que a carga imposta em cada um dos servidores da rede é sempre inferior a um porque, caso isso não aconteça, o sistema é instável e consequentemente o tamanho de alguma das filas do sistema vai crescer para infinito.

Nem sempre se consegue calcular a carga imposta por um buffer a um servidor ou ao sistema. Há casos, como o da Figura 2.1, em que não é possível saber que percentagem dos clientes do buffer 1 vai para que servidor, ou seja, não se sabe a taxa de chegada efectiva de clientes vindos do buffer 1 a cada um dos servidores.

Actualmente a comunidade científica acredita que a condição 2.1 é necessária mas não suficiente para que a rede seja estável, excepto para o caso das redes de Jackson (entre outras), ou seja, redes em que cada servidor processe um só tipo de clientes (vários buffers implicam vários tipos de clientes). Neste contexto, um sistema é considerado estável desde que o tamanho dos buffers que lhe estão associados se mantenha limitado, em termos do seu valor esperado.

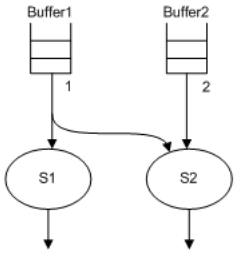


Figura 2.1: Exemplo em que é impossível calcular a carga nos servidores.

Para se poder calcular a carga em cada um dos servidores é preciso saber as taxas de chegada a todos os buffers, internos e externos. Os externos são definidos inicialmente. Para os internos, e uma vez que nesta tese só se estudam redes com routing determinístico em que não existe nem merge nem split, a sua taxa de chegada vai ser igual à taxa de chegada do buffer externo que lhe deu origem.

2.1.3 Redes uniclasse e multiclasse

Existem dois tipos de redes de filas de espera distintas. Estes dois tipos são os seguintes:

- Uniclasse: quando uma rede contém servidores que apenas processam um buffer significa que cada servidor só vê um tipo de cliente, para ele não há distinção de quem serve, uma vez que vêm todos do mesmo buffer;
- Multiclasse: quando existem servidores que processam mais do que um buffer, mesmo que por coincidência a taxa de processamento para cada um desses buffers seja a mesma, significa que os clientes são distinguíveis por alguma razão (podem, por exemplo, encontrar-se em fases diferentes de processamento).

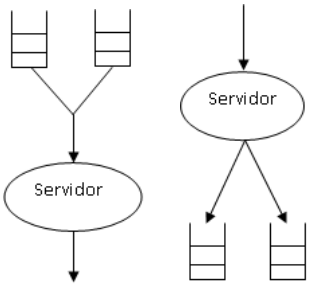
2.2 Caracterização de uma Rede de Actividades

O conceito de redes de filas de espera é simples e com poucas possibilidades de variantes. Com o objectivo de criar um modelo genérico e útil para mais tipos de aplicações do que os habituais (serviços como os que temos habitualmente no nosso dia-a-dia, os correios, serviços de saúde, lojas do cidadão, etc) Harrison, [6, 2], desenvolveu o conceito de rede de actividades. Criou-se então um modelo que abarca uma série de outras configurações passíveis de serem utilizadas noutros contextos como os de manufactura.

Para Harrison cada 'tipo' de serviço passa a ser denominado de Actividade. Cada actividade pode incluir qualquer uma das seguintes configurações, ver Figura 2.2:

- Merge: significa que uma actividade é executada consumindo mais do que uma unidade de material, podendo este ser do mesmo tipo ou de tipos diferentes (diferentes buffers pressupõe diferentes tipos de materiais, caso contrário não faria sentido a distinção);

- Split: a actividade independentemente de ter ou não merge, pode gerar mais do que uma unidade de saída, sendo que nesse caso o que sai pode ser do mesmo tipo ou tipos diferentes. Isto significa que uma só actividade pode ter mais do que um buffer de saída.



(a) Merge. (b) Split.

Figura 2.2: Exemplo de configurações possíveis.

De notar que, quando uma actividade tem uma entrada com uma quantidade diferente de um, podemos estar a falar no contexto de clientes (pessoas) ou de material. Estes dois termos vão por isso ser utilizados consoante o contexto embora no fundo tenham o mesmo significado.

Harrison modela a rede através de vários parâmetros. Cada rede é constituída por S servidores distintos que consomem e produzem unidades de material M através de N actividades diferentes.

A rede caracteriza-se então da seguinte forma:

A matriz A relaciona as actividades com os respectivos servidores. A_{ij} é a taxa de processamento do servidor *i* para realizar a actividade *j*.

Exemplo:

$$A = \begin{bmatrix} 1.3 & 1.1 & 0 & 0 \\ 0 & 1.5 & 0 & 0 \\ 0 & 0 & 1.4 & 1 \end{bmatrix}$$

A matriz R por sua vez traduz a quantidade de material necessária para cada actividade. Considera-se que R_{ij} é a quantidade de material consumida do buffer *i* pela actividade *j* quando este valor é positivo. Quando é negativo significa que a actividade em questão produz material para esse buffer.

Exemplo:

$$R = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}$$

O vector *L* traduz simplesmente as taxas de chegada do exterior λ a cada um dos buffers sendo que se um buffer não tiver chegadas do exterior essa taxa é zero.

Exemplo:

$$L = \begin{bmatrix} 1.1 & 0.8 & 0 & 0 \end{bmatrix}$$

Finalmente o vector Q dá a informação de quantos servidores do mesmo tipo existem.

Exemplo:

$$Q = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$$

Ou seja, neste caso só existe um servidor de cada tipo. Mas se por outro lado se $Q = \begin{bmatrix} 2 & 1 & 1 \end{bmatrix}$, significa que o servidor 1 tem um 'gémeo' que faz exactamente as mesmas actividades e as processa à mesma taxa. Deste modo evita-se a replicação de linhas na matriz *A*. Assim, a rede constituída pelos exemplos anteriores é a da Figura 2.3.

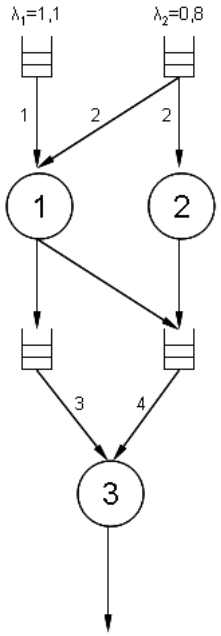


Figura 2.3: Rede constituída pelos exemplos anteriores.

2.2.1 Observações a este tipo de representação

Esta representação tem algumas vantagens e também algumas limitações importantes. A principal limitação deve-se ao facto de só servir para routing estático. Não permite que aconteça o caso em que o resultado de uma actividade vá para um ou outro buffer consoante o tamanho da fila dos mesmos. Este ou outros critérios que impliquem dinamismo necessitariam uma modelação diferente.

Harrison nos seus artigos não refere nem merge nem split. No entanto, a interpretação proposta nesta tese, serve perfeitamente para englobar estas duas opções de configuração. Uma actividade pode precisar de mais do que uma unidade de material e/ou de mais do que um buffer para ser realizada e a sua saída também pode ser superior a uma unidade.

Há também duas hipóteses possíveis para a interpretação da matriz A. A primeira, e a que foi utilizada neste trabalho foi a de que, quando uma coluna da matriz (que corresponde a uma actividade) tem mais do que uma posição diferente de zero significa que é processada em paralelo por mais do que um servidor, ou seja, há mais do que um servidor com essa actividade em comum, no entanto, cada um pode processá-la a uma taxa diferente.

J. G. Dai utiliza num dos seus artigos, [7], a representação de Harrison. No entanto, a sua interpretação foi ligeiramente diferente, uma vez que este ignora sempre os buffers de chegada ao sistema e Harrison não. Dai defende que a representação da matriz A serve para representar 'team work'. Neste contexto 'team work' significa que uma actividade é realizada em conjunto por mais do que um servidor simultaneamente. A modelação proposta por Harrison, nos artigos já citados, não permite em simultâneo a representação de 'team work' e a representação de situações em que servidores diferentes executam de forma independente a(s) mesma(s) actividade(s). Por isso, só podendo ter uma destas possíveis representações, nesta tese adoptou-se a ideia de que quando uma actividade tem mais do que um servidor, estes a processam de forma independente. Olhando para a Figura 2.3 conseguem-se ilustrar estes dois casos. No primeiro se por alguma razão o servidor 2 avaria os clientes do buffer 2 deixam de poder ser processados. E no segundo os clientes continuam a ser processados pelo buffer 1 uma vez que os dois servidores são totalmente independentes.

Capítulo 3

Simulador

Neste capítulo explica-se o funcionamento do simulador construído para simular redes de actividades com as propriedades descritas no capítulo 2. Na secção 3.1 descreve-se o funcionamento geral do simulador, mais concretamente, de que tipo de simulação se trata, do modo como os diferentes constituintes da rede se relacionam, das leis aleatórias que regem os acontecimentos e dos procedimentos da simulação. Na secção 3.2 descreve-se como decorre a simulação propriamente dita. Na secção 3.3 explica-se como deve ser o ficheiro de entrada. O modo de funcionamento da interface com o utilizador e as suas possibilidades de configuração são explicadas na secção 3.4. Finalmente, a secção 3.5 apresenta o modo como são escritos os ficheiros de saída para se poder fazer processamento posterior dos dados.

3.1 Funcionamento Geral

Este pacote foi desenvolvido de modo a poder processar as redes de actividades previamente descritas, excepto para o caso do 'team work', uma vez que, como foi visto no capítulo anterior, este não é compatível com o facto de servidores distintos poderem ter actividades em comum. A linguagem de programação escolhida foi uma linguagem orientada a objectos, neste caso o JAVA, [8], devido à facilidade de criar modelos discretos, em que todos os acontecimentos podem ser descritos por eventos e suas sequências e também pela facilidade em fazer movimentar os materiais (clientes) dentro da rede desde que entram no sistema até que saem.

Como já foi visto, uma rede caracteriza-se por materiais, buffers, servidores e actividades. Estas são por isso, as classes que modelam a rede pretendida e estão relacionadas como ilustrado na Figura 3.1. Tal como se pode observar, existem relações que são dirigidas, o que significa que são uni direccionais.

Uma vez que se trata de simulação de redes de actividades a actividade é a classe central no que respeita à modelação da rede. Esta pode ser processada por um ou mais servidores (1..*) e por isso tem associada uma lista de servidores (listaServ). Analogamente aos servidores, também pode ter mais do que um buffer de entrada, no caso de ter merge, e tem de ter pelo menos um buffer de saída, dependendo se a actividade tem split ou, por exemplo, processa mais do que uma unidade de material em simultâneo. Por sua vez um buffer contém uma lista de clientes, bem como cada servidor.

Na estrutura da rede propriamente dita, a lista de servidores, buffers de entrada e de saída têm um tamanho fixo durante toda a simulação e dependem da topologia da rede que se está a simular. As únicas listas cujo tamanho é variável são as duas listas de clientes. Os clientes desde que entram na rede até que saem estão num buffer à espera de serem processados (listaClientes), ou estão a ser processados por um servidor e nesse caso, estão na sua lista de atendimento (listaAtendimento).

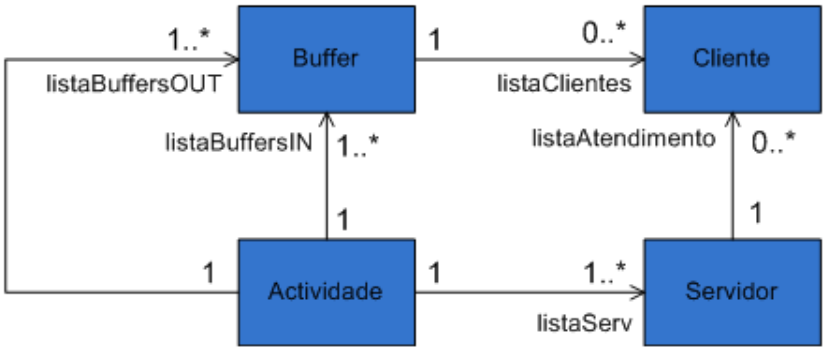


Figura 3.1: Relações entre classes que modelam a rede.

Uma vez que se trata de simulação estocástica discreta, começou por ser essencial identificar os eventos que afectam o sistema a modelar. Para isso, foi necessário dividi-los por categorias, de modo a que pudessem ser distinguíveis e colocar-lhes um atributo de tempo. Esse atributo marca o instante em que se dá o evento. De seguida foi preciso definir as leis aleatórias que regem a ocorrência dos eventos, definir os procedimentos para simular a observação das variáveis aleatórias em causa e finalmente simular os eventos.

Para simular redes de actividades, os eventos que desencadeiam acções são os seguintes:

1. Evento chegada: chegada do exterior de uma unidade de material a um buffer (pode fazer com que se inicie uma nova actividade);
2. Evento fim de serviço: termina o processamento de uma actividade, após o que ou começa nova actividade ou fica inactivo, caso não tenha mais nada para fazer;
3. Evento avaria: avaria de um servidor (que fica inactivo até que seja reparado);
4. Evento fim de reparação: servidor torna a ficar activo, iniciando uma nova actividade caso possa;

5. Evento desbloqueia buffer: evento utilizado unicamente quando o mecanismo de bloqueio de buffers está activo (utilizado para testar algumas politicas de sequenciamento).

Estes eventos estão representados na Figura 3.2.

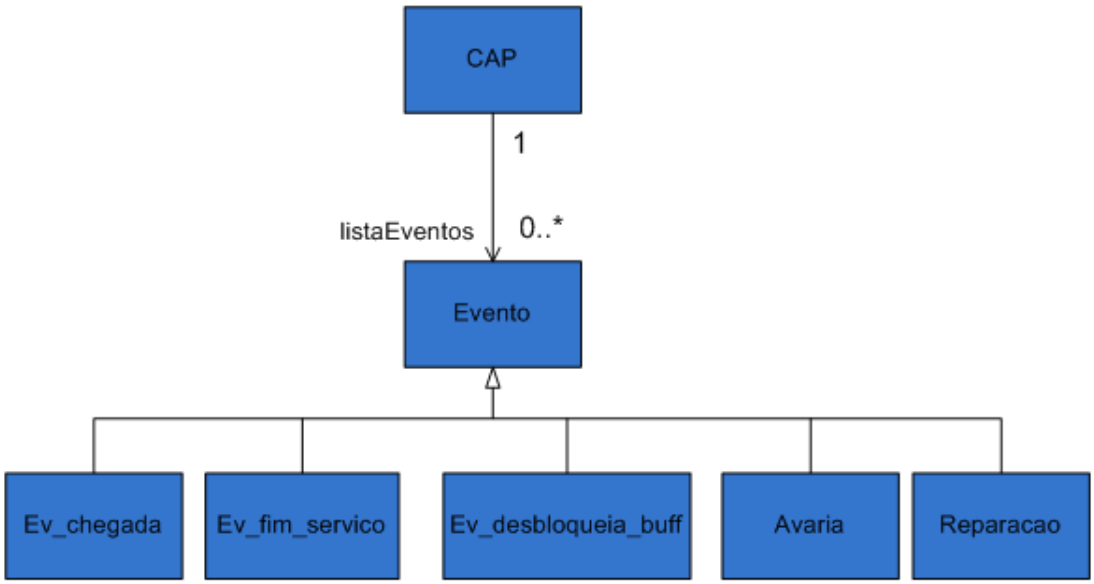


Figura 3.2: Eventos existentes no simulador.

Neste caso a classe central é a CAP (Cadeia de Acontecimentos Pendentes), que é uma lista de eventos pendentes (futuros). A classe evento contém alguns métodos que são herdados por todas as classes dos eventos. Tal como acontece com as listas de clientes, a CAP pode estar vazia. No entanto se isso acontecer, significa que ou os eventos iniciais ainda não foram gerados ou a simulação terminou.

3.1.1 Geração de variáveis aleatórias

A geração de uma realização de uma variável aleatória exponencial de média m é feita por intermédio da inversa da função de distribuição tal como descrito na Expressão 3.1, em que $random$ traduz o número aleatório que foi gerado a partir de uma distribuição uniforme entre 0 e 1.

$$-m \times log(1 - random) \tag{3.1}$$

Chegadas: admite-se que o tempo entre chegadas consecutivas a um buffer k é uma variável aleatória com distribuição exponencial de valor médio $1/\lambda_k$;

Fim de serviço: admite-se que o tempo que um servidor i demora a realizar uma actividade j é uma variável aleatória com distribuição exponencial de valor médio $1/\mu_i^j$;

Desbloqueia buffer: este evento é o único que não se rege por nenhuma variável aleatória do tipo Random, é gerada pelo próprio simulador;

Avaria e fim de reparação: estes eventos não foram ainda implementados, não tendo sido por isso definida a variável aleatória que os rege. No entanto, a estrutura que permite a sua inclusão está toda implementada.

O Java disponibiliza para este efeito a classe Random, já com alguns métodos que facilitam a geração de números aleatórios para a distribuição pretendida bem como para algumas outras.

Dado que este trabalho tem como objectivo a estabilização de redes e melhoramento dos tempos de ciclo, era importante garantir que, para uma mesma rede, alterando as políticas de sequenciamento, os seus resultados fossem comparáveis. Ou seja, que as diferenças observadas se devessem unicamente à diferença entre as políticas de sequenciamento e não ao facto de uma política ter tido mais sorte do que a outra na ordem dos números aleatórios gerados.

Para exemplificar esta situação suponhamos agora que queremos comparar a política de sequenciamento FIFO com outra política diferente. Existem dois sistemas com apenas um servidor tal como na Figura 3.3 e cuja única diferença é a política. Se as distribuições do processo de chegada e do tempo de serviço forem as mesmas para ambos os sistemas, uma correcta utilização dos Random's implica que os dois sistemas vêem os mesmos clientes a chegar nos mesmos tempos e os tempos de serviço para cada um desses clientes também vão ser os mesmos, independentemente da ordem pela qual são processados. Para que estas condições sejam cumpridas, é preciso que os processos sejam independentes uns dos outros, [9], o que torna necessário que por cada processo exista um objecto do tipo Random.

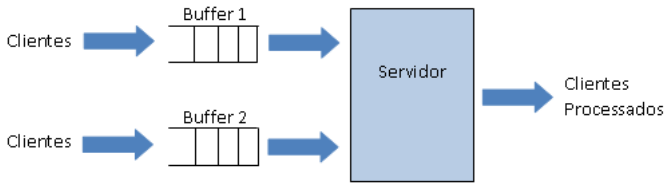


Figura 3.3: Exemplo de uma rede.

Se existir só um objecto do tipo Random significa que a sequência dos números gerados vai ser sempre a mesma. No entanto, a duração do evento gerado vai ser diferente se a ordem dos eventos gerados for diferente. Isto porque a duração do evento depende não só do aleatório gerado ($random$) mas também do valor médio da variável aleatória (m).

Tendo em conta que se tratam de redes de actividades, e que a ordem com que são processadas depende precisamente da política de sequenciamento escolhida para cada servidor, a simulação vai ter um objecto do tipo Random por cada actividade, mais um para o processo de chegada ao sistema.

Existe um método chamado expRandom2 para a geração dos intervalos entre chegadas e um outro expRandom3 para a geração dos tempos em que os eventos fim de serviço vão ocorrer. Existe ainda um outro método expRandom cuja razão de existir é explicada na secção seguinte.

Se no futuro se pretender utilizar outras distribuições, basta alterar os métodos existentes para que possam ter mais do que uma opção disponível. Assim será possível gerir a geração das diferentes variáveis aleatórias de acordo com a distribuições pretendidas, adicionando à interface a opção de escolha.

3.1.2 Definição dos procedimentos a simular

Depois de definidos os eventos é preciso estabelecer o que acontece em cada um deles. Sempre que a simulação começa é preciso inserir na CAP (Cadeia de Acontecimentos Pendentes) um evento chegada por cada buffer com chegadas do exterior. Isto porque o simulador começa sempre com as filas vazias. Se o objectivo for fazer o estudo já com pessoas na fila então isso resolve-se quando se processarem os dados dos ficheiros de saída da simulação, ignorando os primeiros *x* clientes do sistema).

Uma definição importante para compreender os procedimentos da simulação é a de actividade factível. Uma actividade é considerada factível quando todo o material necessário para que esta seja efectuada existe no(s) buffer(s), pelo menos um servidor está disponível para a executar e, no caso de o mecanismo de bloqueio a buffers estar activado, os buffers em questão estarem desbloqueados.

- **Evento chegada:** Sempre que ocorre uma chegada do exterior a um buffer, o simulador verifica se alguma actividade fica factível. Se isso acontecer, calcula a duração dessa mesma actividade, o tempo em que esta vai terminar e insere na CAP o evento fim de serviço correspondente. Ao mesmo tempo coloca o material que é retirado do(s) buffer(s) numa lista temporária do servidor. Caso a chegada não torne nenhuma actividade factível, o material fica no buffer à espera de poder ser processado. Sempre que ocorre um evento deste tipo é gerado um novo evento para a próxima chegada a este buffer.
- **Evento fim de serviço:** Quando é retirado da CAP um evento fim de serviço, o material sai do servidor e é colocado no buffer de saída de acordo com a especificação da rede. Esta chegada interna a um buffer tem como consequência o mesmo procedimento que uma chegada exterior, com a excepção óbvia da geração do novo evento chegada. Por sua vez, o servidor que acabou de ficar livre vai ver se pode iniciar mais alguma actividade. Se sim, inicia-a e insere o novo evento fim de serviço correspondente. Caso contrário fica inactivo à espera que haja outra chegada a um buffer. Caso o bloqueio a buffers esteja activado, o evento fim de serviço vai sempre verificar se o deve bloquear ou não. O mecanismo de bloqueio a buffers é explicado em detalhe no Capítulo 4.

- **Evento desbloqueia buffer:** Neste evento o buffer torna a ficar desbloqueado e, caso este tenha clientes, verifica se se pode iniciar uma nova actividade. Se isso acontecer, o procedimento é, mais uma vez, semelhante a uma chegada do exterior. Insere-se o evento fim de serviço na CAP, retira-se o material do(s) buffer(s) respectivo(s) e coloca-se na lista temporária do servidor.

Uma chegada de um cliente a um buffer pode fazer com que mais do que uma actividade fique factível. No caso de essa(s) actividade(s) implicar(em) só um servidor, esse servidor vai decidir que actividade processa com base na política de sequenciamento. Mas se essa(s) actividade(s) implicar(em) mais do que um servidor (significa que existe mais do que um servidor livre nesse momento), uma vez que a política de sequenciamento é a nível do servidor, então escolhe-se um servidor aleatoriamente e este decide que actividade é que vai ser efectuada de acordo com a sua política. Isto porque não estão implementadas políticas de routing.

Para que o simulador fosse o mais versátil possível, foram implementadas várias políticas de sequenciamento. É possível que cada servidor tenha a sua própria política, permitindo assim muitas mais configurações para uma mesma rede. As políticas implementadas são as representadas na Tabela 3.1. A política FIFO2 é uma política local mas adaptada ao conceito de actividade. Em vez de contar o tempo de chegada a um buffer o que conta é o tempo em que as actividades se tornam factíveis, ou seja, abrange a possibilidade de merge. As políticas FSVCT e FSMCT são as políticas de *least slack* implementadas e Threshold foi uma política desenvolvida por Zita Fernandes na sua tese, [10]. O artigo [11] apresenta uma síntese do simulador que se descreve nesta tese, em conjunto com alguns dos resultados obtidos para a política Threshold.

Tabela 3.1: Políticas implementadas

Número da política	Política
1	FIFO2
2	FSVCT
3	FSMCT
4	μc Rule
5	Threshold
6	LBFS
7	FIFO

3.2 Simulação dos eventos

Após se terem definido todas as leis aleatórias, os eventos existentes e os respectivos procedimentos é preciso simular esses eventos. Cada instância do simulador vai então ter uma CAP (Cadeia de Acontecimentos Pendentes) que não é mais do que uma lista de eventos (futuros) ordenados pelo tempo em que devem ocorrer.

Durante a simulação vão-se retirando um a um os eventos, que só são executados se ainda não se tiver atingido o critério de paragem escolhido (Secção 3.4.3). A execução dos eventos é que faz com que novos eventos sejam gerados. Um evento chegada gera sempre outro evento chegada, um evento fim de serviço e um desbloqueia buffer podem ou não gerar um evento fim de serviço, um evento avaria gera sempre outro evento avaria e, finalmente, um evento reparação pode ou não gerar um evento fim de serviço. A simulação termina quando já não existirem mais eventos na CAP.

Com base na discussão anterior, o UML (Unified Modeling Language) final é o da Figura 3.4.

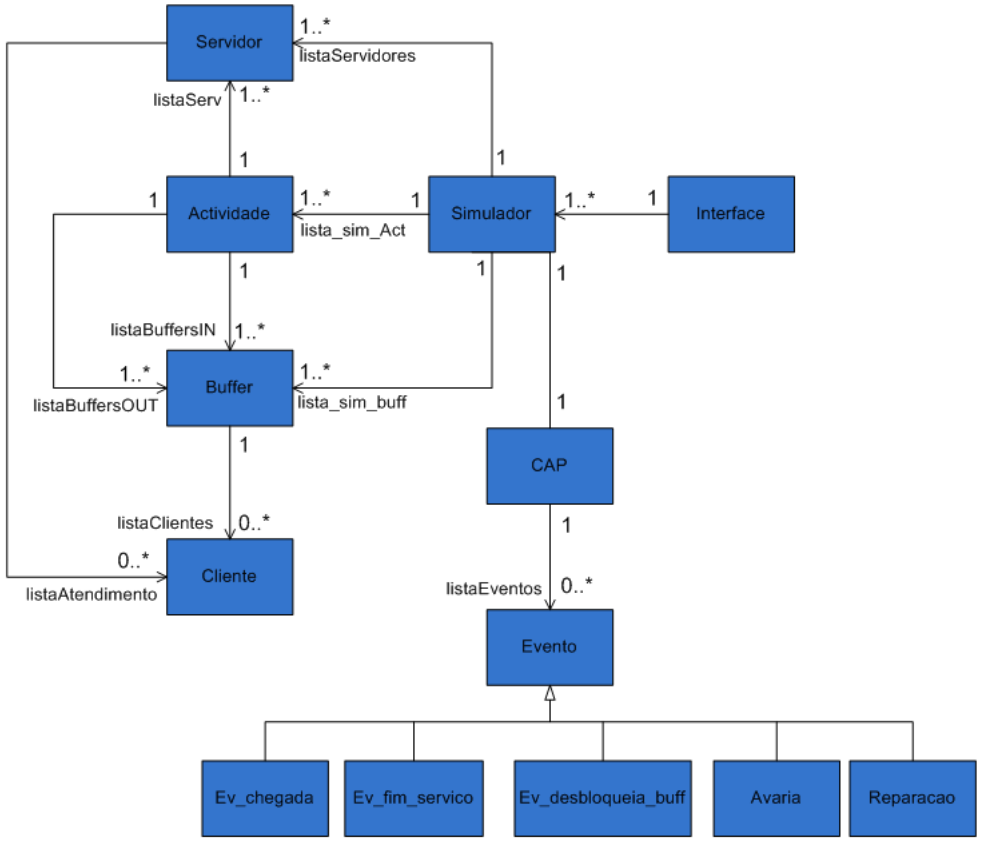


Figura 3.4: UML com todas as classes implementadas.

3.3 Ficheiro de Entrada

Para simular as redes é preciso que sejam introduzidos os parâmetros da rede, tal como descrito por Harrison. O ficheiro de entrada para a rede exemplificada na Figura 2.3 tem o formato ilustrado na Figura 3.5. Como se pode observar, além dos parâmetros da rede existem também outros parâmetros que ainda não foram referidos. Todos estes estão relacionados com as políticas de sequenciamento implementadas, podendo ou não ser necessários consoante as políticas seleccionadas para a simulação.

```
servidores 3;
buffers 4;
actividades 4;
A=[1.3 1.1 0 0;
 0 1.5 0 0;
 0 0 1.4 1];
R=[1 0 0 0;
 0 1 0 0;
 -1 0 1 0;
 0 -1 0 1];
L=[1.1 0.8 0 0];
Q=[1 1 1];
csi=[0 0 0 0];
custos=[];
politicas=[7 7 7];
tipo bloqueio=1;
delta=[0 0 0 0];
```

Figura 3.5: Exemplo de um ficheiro de entrada.

Embora inicialmente possa parecer uma boa ideia separar o ficheiro de entrada em dois, um para as definições da rede propriamente dita e um outro para restantes variáveis, na realidade tal não é possível, uma vez que o simulador é muito flexível a nível de configuração. As razões são as seguintes:

- O vector ξ corresponde ao tempo médio de ciclo restante e está relacionado com as políticas FSMCT e FSVCT e a sua dimensão depende do número de buffers da rede;
- O vector de custos está relacionado com as políticas μC e Threshold estudadas em [10];
- O vector políticas indica a política de sequenciamento de cada um dos servidores da rede e a sua dimensão depende do número de servidores da rede;
- O tipo de bloqueio indica qual o bloqueio a ser utilizado, 1 ou 2. Este mecanismo é explicado na secção 4.2;
- O vector δ é utilizado quando o bloqueio é do tipo 2 e a sua dimensão também depende do número de buffers da rede, uma vez que este permite activar bloqueio a buffers apenas nos especificados. Ou seja, só quando uma posição for diferente de zero é que esse buffer pode ser bloqueado.

As dimensões dos vectores estão, como se pode ver, directamente relacionados com a topologia da rede, pelo que não podem ser separados.

A razão porque se escolheu que estes dados fossem lidos de um ficheiro de entrada e não de outro tipo de interface está relacionada com razões práticas. Se uma rede tiver por exemplo, 24 servidores distintos (sem contar com os 'gémeos' pois estes não entram na matriz A), 172 actividades e 172 buffers estamos a falar de uma matriz A de 4128 entradas e de uma matriz R de 29584 entradas e os restantes vectores também têm 172 posições. Desde modo basta criar o ficheiro só uma vez que depois pequenas alterações são muito fáceis de introduzir.

3.4 Interface

A interface com o simulador é simplesmente um conjunto de opções que não fazem sentido que sejam definidas pelo ficheiro de entrada. Estas opções são as seguintes:

1. Escolha do ficheiro da rede que se pretende simular;
2. Número de simulações pretendidas;
3. Critério de paragem da simulação;
4. Bloqueio de buffers activo ou inactivo.

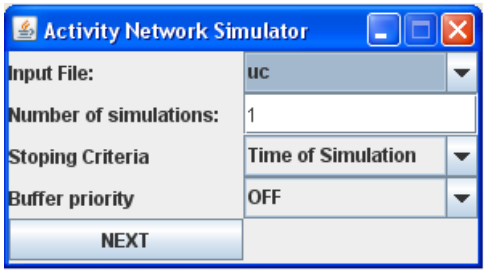


Figura 3.6: Aspecto da interface.

3.4.1 Escolha do ficheiro da rede

Existem várias redes disponíveis para fazer testes. No entanto, caso se pretenda adicionar mais algum, é extremamente fácil inseri-lo na lista respectiva. A razão pela qual só é possível optar a partir de um menu de ficheiros tem como objectivo garantir que se escolhem sempre ficheiros que existam realmente.

Os ficheiros de entrada disponíveis neste momento são os representados na Figura 3.7

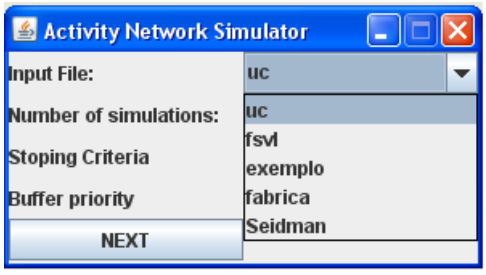


Figura 3.7: Escolha dos ficheiros de entrada na interface.

3.4.2 Escolha do número de simulações

O número de simulações pretendidas está unicamente relacionado com o facto de se tratar de simulação estocástica. À partida os comportamentos devem ser consistentes e muito semelhantes. A possibilidade de replicar a simulação para um mesmo conjunto de parâmetros de entrada, serve o propósito de se calcularem intervalos de confiança para as medidas de desempenho utilizadas.

3.4.3 Critério de paragem da simulação

É extremamente importante que o critério de paragem da simulação seja bem escolhido. Existem três critérios possíveis, tal como se pode ver pela Figura 3.8.

- Tempo de simulação: duração da simulação;
- Intervalos de regeneração: um intervalo de regeneração é atingido quando num dado momento da simulação não existe ninguém na rede. Ou seja, os buffers estão vazios e os servidores estão todos livres. Dependendo da rede, estes intervalos é que definem a duração da simulação. Pode haver casos em que uma vez iniciada a simulação, a rede nunca mais torna a ficar vazia, ou por outro lado, podem ser atingidos com muita facilidade;
- Número de clientes: número de clientes que saiu do sistema.

A escolha deve depender do tipo de rede e da carga da mesma. Se se tratar de uma rede com uma carga bastante elevada e/ou bastante grande deve-se escolher o número de clientes que saem do sistema ou tempo de simulação (dependendo do tipo de processamento que se pretende fazer depois da simulação), pois os intervalos de regeneração dificilmente serão atingidos.

Por sua vez, se a carga for excepcionalmente pequena e/ou a rede também o for, então o mais adequado provavelmente é mesmo a opção dos intervalos de regeneração.

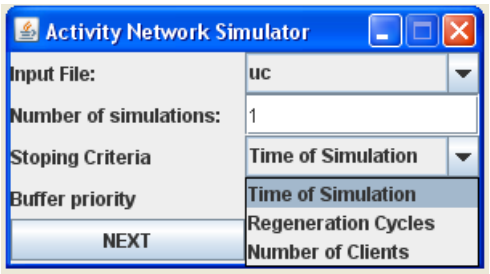


Figura 3.8: Escolha dos critérios de paragem na interface.

É preciso ver que o número de clientes que saíram do sistema pode não ser igual ao número de clientes que entraram no caso de haver merge e/ou split. Neste caso os clientes que saem podem ser

mais, menos ou em igual número, dependendo da configuração da rede e da quantidade de material necessária para realizar cada actividade.

Seja qual for o critério de paragem escolhido, a interface pede sempre para especificar a quantidade. Sejam eles, o número de intervalos de regeneração, a duração da simulação ou o número de clientes que sai do sistema.

3.4.4 Bloqueio de Buffers Activo ou Inactivo

Esta opção apenas faz com que o controlador fique activo ou inactivo. O ficheiro de entrada deve especificar o tipo de bloqueio que deve ser activo (ver Figura 3.5). Tal como descrito no Capítulo 4, existem dois métodos para calcular o parâmetro referenciado em 4.1. Se cada actividade for processada apenas por um servidor então o ficheiro de entrada deve especificar 1. Por outro lado, se houver pelo menos uma actividade que seja processada por mais do que um servidor, então o tipo de bloqueio escolhido deve ser o 2.

3.5 Ficheiros de Saída

Uma vez que o simulador tem como objectivo simular redes de actividades e consequentemente retirar dados que permitam o estudo dos seus parâmetros, existem vários ficheiros de saída que fornecem os dados relativos a todos os eventos ocorridos durante a simulação. Estes dados vão ser sujeitos a pós processamento para se calcular tempos de permanência no sistema, tempos de ciclo e tempos de ciclo restantes (quando um cliente já não se encontra no buffer inicial), entre outros possíveis. Estes dados também permitem estudar a estabilidade de redes e mecanismos de estabilização. Nesta tese o pós processamento foi efectuado em Matlab.

Os ficheiros têm sempre um nome do tipo **_i* em que *i* significa o número da simulação que lhe foi associada, e *** é o nome do ficheiro em questão. Se só se pedir uma simulação então só vai ser gerado um conjunto de ficheiros, se se pedir duas então cria dois conjuntos, etc. Os ficheiros são os seguintes:

- 'balanco_i': Este ficheiro é escrito sempre que alguém sai do sistema e permite fazer o balanço entre o número de clientes que entraram e saíram do sistema;

    1. Número de clientes que entraram no sistema;
    2. Número de clientes que saíram do sistema;
    3. Tempo em que se escreveu no ficheiro (tempo de saída do cliente).

- 'buffers_i': Este ficheiro é escrito só no final da simulação e indica quanto tempo esteve cada buffer bloqueado (se o bloqueio a buffers não foi activado esse tempo é sempre zero);

    1. ID do buffer;
    2. Tempo total de bloqueio.

- 'duracao_i': Aqui é indicada simplesmente a duração da simulação, uma vez que esta nem sempre é especificada como critério de paragem;

- 'resultados_i': Através desta tabela é possível fazer os cálculos dos tempos de ciclo para cada um dos buffers e dos tempos de permanência no sistema. A escrita deste ficheiro é feita sempre que se termina um serviço;

    1. Id do buffer de chegada;
    2. N-ésima chegada a esse buffer;
    3. Tempo de chegada ao sistema;
    4. Tempo de saída do buffer;
    5. Id do buffer respectivo de onde saiu;
    6. Tempo de saída do sistema (quando este campo é preenchido os dois itens anteriores estão a zero).

- 'servers_i': Este ficheiro é escrito no final da simulação. Permite saber quanto tempo esteve cada servidor inactivo e porquê (se forçado ou se simplesmente não tinha nada para fazer);

    1. Id do servidor;
    2. Id de quem é cópia (no caso de ser 'gémeo' de algum outro servidor);
    3. Duração de inactividade passiva;
    4. Duração de inactividade activa.

- 'total_i': Este ficheiro é escrito sempre que se termina um serviço e permite saber qual o tamanho das filas associadas a cada servidor respectivo.

    1. Id do servidor que terminou o serviço;
    2. Existem tantas mais colunas como o número de buffers existentes no sistema, o valor é zero se o servidor com o id da coluna 1) não servir esse buffer.

Capítulo 4

Controlador

O objectivo desde capítulo é introduzir o conceito de inactividade activa, explicar como esta pode ser benéfica para o melhoramento do desempenho de uma rede e a razão de se ter desenvolvido um controlador que introduz esta inactividade. A primeira secção explica estes conceitos e na secção 4.2 descreve-se o funcionamento do controlador implementado em detalhe.

4.1 Introdução

No passado acreditava-se que a ordem com que os clientes eram processados afectava exclusivamente o desempenho do sistema, desde que a capacidade de serviço fosse superior à carga imposta pelo processo de chegada, $\rho_i < 1$, em que i identifica o servidor em questão.

Como se pode ver pelo trabalho desenvolvido por Seidman, [3], a realidade não é exactamente assim quando se está a falar de redes multi-classe. A ordem com que os clientes são atendidos pode afectar a estabilidade do sistema mesmo se as políticas adoptadas forem non-idling.

A maior parte da comunidade continua concentrada em encontrar políticas de sequenciamento non-idling. No entanto, acreditamos que qualquer rede pode ser estável quando a condição de tráfego se verifica. Nesta perspectiva, uma rede de filas de espera é considerada estável desde que exista pelo menos uma política que a estabilize.

Existem políticas que fazem claramente com que o sistema fique instável. Por sua vez, políticas como as de *least slack* garantem a estabilidade da rede, [12].

No entanto as políticas de decisão local têm uma grande vantagem face às de *Least Slack*. Estas políticas não precisam de informação do sistema, o que faz com que sejam muito mais fáceis de implementar e impliquem um fluxo de informação muito pequeno.

No caso de uma rede com políticas non-idling, um servidor, não tendo clientes à espera de serem processados, fica inactivo. Essa inactividade vai-se manter até que chegue alguém a um buffer que esse servidor processe. Ter clientes no buffer e obrigar o servidor a manter-se inactivo não parece à primeira vista que sirva para alguma coisa, antes pelo contrário, parece só poder piorar o desempenho e contribuir para a instabilidade do sistema. Mas a realidade é que quando se está a ver a perspectiva de um só servidor não se tem a percepção do que isso pode afectar o resto da rede, fenómeno que acontece quando a política de sequenciamento é local.

Um serviço mais longo do que o normal, uma perturbação no processo de chegada, etc, podem fazer com que a rede perca o seu equilíbrio. Mesmo no caso de os tempos serem determinísticos uma avaria no servidor pode ser o suficiente.

A solução aqui proposta não é mais do que um mecanismo de controlo que, em alguns momentos, introduz inactividade nos servidores com o objectivo de reduzir tempo médio de permanência no sistema (TMPS). Inactividade activa significa que, esporadicamente, o servidor tem a capacidade de bloquear um determinado buffer mesmo que, em alguns casos, isso implique permanecer inactivo na presença de clientes em espera para serem processados. O que isto significa é que um servidor, mesmo tendo uma política non-idling deixa de conseguir 'ver' os buffers que estão bloqueados, o que faz com que, se todos os outros buffers que ele serve estiverem vazios, o servidor fique inactivo. Por sua vez, bloquear um buffer pode fazer com que o servidor fique disponível para processar outros buffers que de outro modo não seriam processados tão rapidamente.

O controlador introduzido foi desenvolvido de modo a conjugar-se com qualquer política de sequenciamento escolhida, ou seja, a política em si não é alterada. Este aspecto traz consigo várias vantagens. Em primeiro lugar, a implementação das políticas mantém-se a mesma seja ela qual for. Em segundo, o controlador para tomar as suas decisões não precisa de olhar para toda a rede e, a nível de simulação, pode ser um mecanismo que é activado ou desactivado consoante o objectivo do utilizador. Estes aspectos são vantajosos tanto a nível estrutural como a nível de implementação.

Com este controlador vai-se tentar estabilizar e mesmo melhorar o desempenho de várias políticas.

É crucial que o controlador seja bem definido, caso contrário o desempenho piora efectivamente. Se o tempo de inactividade for introduzido na altura temporal errada ou em demasia, o tempo de permanência no sistema aumenta brutalmente, fazendo o efeito contrário do pretendido.

Para compreender o funcionamento do controlador é importante perceber a razão pela qual a introdução de inactividade é vantajosa. Tomemos por exemplo uma rede com apenas uma classe de clientes, ou seja, uma rede em que apenas existe um buffer com chegadas do exterior, mas em que o número de visitas desses clientes a pelo menos um dos servidores da rede é superior a um. Considerando

o exemplo da Figura 4.1, se, por acaso, num curto intervalo de tempo chegarem muitos clientes ao sistema, o servidor 1 fica momentaneamente com muito serviço para fazer. Se essa fila for prioritária, o que vai acontecer é que esses clientes vão ser atendidos e, por isso, o servidor vai-se manter ocupado e os outros clientes que se encontram no buffer 2 vão ficando à espera. O burst de chegadas vai ser propagado para o buffer 2 e o tamanho desta fila vai crescer. A questão é que se o burst de chegadas for relativamente grande, o tempo que o servidor 1 deixa de processar os clientes que se encontram no buffer 2 pode fazer com que o buffer 3 fique vazio e por isso o servidor 2 fica inactivo. Bloqueando esporadicamente o buffer 1 obriga-se o servidor 1 a ficar disponível para processar clientes do buffer 2, não deixando que o buffer 3 fique vazio e por isso os recursos do sistema sejam melhor aproveitados.

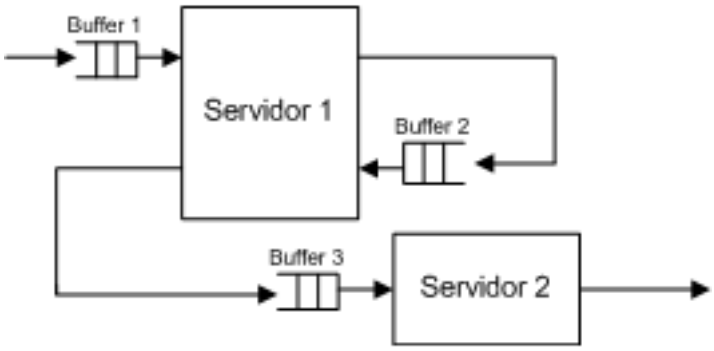


Figura 4.1: Exemplo explicativo da importância de inactividade.

Embora no capítulo 2 tenham sido descritas redes de actividades, o controlador foi desenhado para redes de filas de espera, ou seja, actividades só com um buffer de entrada e um outro de saída, mas, ainda assim, com possibilidade de existência de mais do que um caminho.

4.2 Controlador

Como já foi dito, é extremamente importante que a decisão de bloqueio a um buffer seja bem tomada, caso contrário o resultado piora significativamente. Pretende-se impedir que um servidor dedique demasiado tempo a um buffer, deixando outros acumularem demasiados clientes. É claro que o número de clientes em espera na realidade não é a única coisa importante. Se forem muitos clientes mas estes forem processados a uma taxa muito alta, para o servidor pode significar pouco tempo de trabalho. Por outro lado, mesmo sendo muito poucos clientes, se a taxa a que o servidor os processa for muito baixa, pode significar muito tempo de trabalho.

O que não se quer é que por causa de um servidor estar ocupado a processar clientes de determinado tipo existam buffers que fiquem vazios porque o servidor não tem as quotas de tempo dedicado a cada buffer bem definidas. Se este fenómeno acontecer, pode implicar que determinados servidores parem

porque não têm ninguém para atender.

Tal como já foi referido não é o tempo que um servidor dedica a cada actividade o único factor relevante, mas sim a conjugação do tempo dedicado, com a taxa a que essa actividade é processada. O controlador o que faz é então estabelecer quotas para cada um dos servidores relativamente às actividades respectivas, adequadas à carga que cada uma das actividades impõe no servidor correspondente. Se essas quotas forem ultrapassadas, então esse buffer é bloqueado até que passe tempo o necessário para o desbloquear. Sempre que um servidor termina um serviço vai verificar se ultrapassou a quota para a actividade que terminou de processar.

Se o tempo dedicado, desde o início da simulação, pelo servidor i à actividade j for superior ao tempo actual multiplicado por um parâmetro θ_j^i (Equação 4.1) o buffer que corresponde à actividade j fica bloqueado até que exista um tempo t_1 que verifique a Equação 4.2.

$$t_{dedicado\_ij} > t_{actual} \times \theta_j^i \tag{4.1}$$

$$t_{dedicado\_ij} = t_1 \times \theta_j^i \tag{4.2}$$

Ou seja, o buffer vai ser desbloqueado em t_1.

$$t_1 = \frac{t_{dedicado\_ij}}{\theta_j^i} \tag{4.3}$$

O tamanho da janela do controlador é, por isso, igual ao tamanho da simulação no instante em que termina um serviço, ou seja, cresce com o tempo.

Neste momento falta apenas definir o parâmetro θ_j^i que aparece nas equações anteriores. Existem dois casos que implicam uma resolução distinta. No primeiro, cada actividade tem um só servidor que a processa, o que significa que sabemos exactamente a carga que essa actividade impõe no seu servidor, fazendo com que seja fácil definir a sua quota. Essa quota é obrigatória que esteja disponível, porque caso contrário a sua fila vai crescer indefinidamente. O que se faz é simplesmente dar um pequeno 'extra' e não mais do que isso.

$$\theta_j^i = \underbrace{\frac{\lambda_j}{\mu_{ij}}}_{quota} + \underbrace{\frac{\epsilon \times \frac{\lambda_j}{\mu_{ij}}}{\rho_i}}_{ajuste} \tag{4.4}$$

ρ_i é a carga no servidor i, λ_j a taxa de chegada da actividade j e μ_{ij} a taxa de processamento da actividade i pelo servidor j. A variável ϵ é definida na equação 4.5.

$$\epsilon = \frac{1 - \rho_i}{k} \tag{4.5}$$

É importante garantir que a Equação 4.6 se verifica. Caso contrário, significa que um servidor está a dar, no total, mais do que 100% do seu tempo, o que é tecnicamente impossível. Nesta equação, N corresponde ao número de buffers da rede.

$$\sum_{j=1}^{N} \theta_j^i \times c(i,j) \leq 1 \tag{4.6}$$

$$c(i,j) = \begin{cases} 1 & \text{se } j \text{ é processado por } i \\ 0 & \text{c.c.} \end{cases} \tag{4.7}$$

Relativamente ao $ajuste$, chamando ρ_{ji} à carga que a actividade j impõe ao servidor i, Equação 4.8, chega-se à Equação 4.9, em que, o somatório não é mais do que uma normalização das quotas pré-definidas de cada servidor a cada uma das suas actividades.

$$\rho_{ji} = \frac{\lambda_j}{\mu_{ij}} \tag{4.8}$$

$$\epsilon \times \sum_{j=1}^{N} \frac{\rho_{ji}}{\rho_i} = \epsilon \tag{4.9}$$

Como o total da carga de um servidor mais o somatório dos $ajuste$ dados a cada uma das actividades tem obrigatoriamente que ser inferior ou igual a um, chega-se à Equação 4.10. O valor de ϵ corresponde então ao total dos ajustes dado por um servidor às suas actividades. Essa folga é então repartida pelos várias actividades, razão pela qual k aparece na Equação 4.5.

$$\rho_i + \epsilon \leq 1 \iff \epsilon \leq 1 - \rho_i \tag{4.10}$$

No segundo caso, uma actividade pode ser processada por mais do que um servidor. Ou seja, tal como foi visto no capítulo 2, não é possível calcular a carga que cada actividade impõe a cada um dos servidores que a processa, porque não se sabe que percentagem de clientes vai para que servidor. O que faz com que seja impossível aplicar a equação 4.4. Nesse caso θ_j^i é dado pela expressão seguinte

$$\theta_j^i = q_j - \delta_j. \tag{4.11}$$

Neste caso q_j traduz o número de servidores que processam a actividade j. Deste modo englobam-se os casos em que existem servidores 'gémeos' e os casos em que servidores não 'gémeos' têm actividades em comum. Enquanto que $t\_actual * q_j$ traduz a quota máxima atribuível a uma actividade por todos os servidores que a processam, o que se faz com δ_j é retirar alguma da folga existente para essa actividade. Este valor é lido do ficheiro de entrada, e por isso é o utilizador que o define. De notar que o caso em que uma actividade só é processada por servidores 'gémeos', a carga que esta impõe a cada um dos servidores é calculável (corresponde só a uma divisão na taxa de chegada efectiva a cada servidor: $\frac{\lambda}{s}$ em que s é o número de servidores gémeos).

Analogamente ao trabalho desenvolvido por José Moreira na sua tese, [4], o controlador tem o mesmo tipo de comportamento, mas, existe uma diferença substancial. Nesta tese o tamanho da janela do controlador é igual ao tamanho da simulação, enquanto que no caso da tese do José Moreira a janela do controlador era de tamanho pré-definido, finito e fixo.

Capítulo 5

Resultados Numéricos

Neste capítulo apresentam-se os resultados numéricos obtidos com o simulador e com o respectivo pós processamento dos dados. Na secção 5.1 estudou-se o comportamento da rede apresentada por Seidman, [3], com e sem controlador. Na secção 5.2 apresentam-se os resultados de tentativas de melhoramento de desempenho da rede estudada por Lu *et al.* em [1].

5.1 Estudo da rede de Seidman

Em [3], Seidman estudou o comportamento da rede apresentada na Figura 5.1.

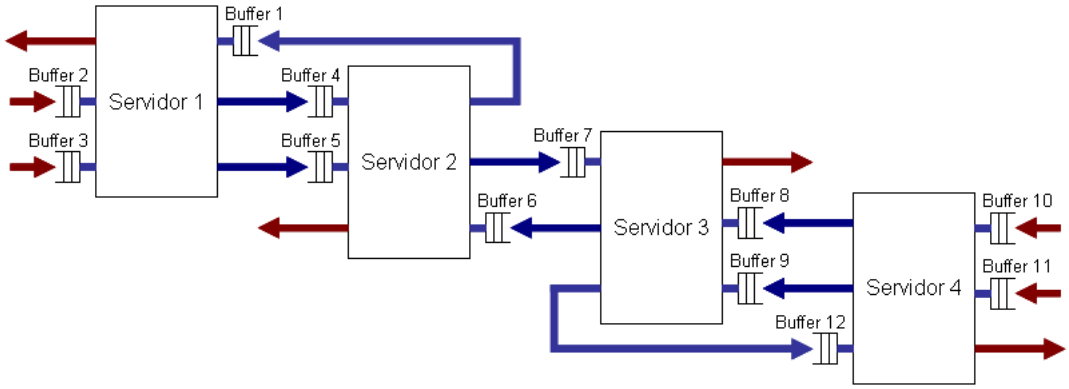


Figura 5.1: Rede estudada por Seidman.

Apesar de os parâmetros que a definem respeitarem a condição de tráfego (5.1), a verdade é que esta rede com uma política de sequenciamento FIFO tem um comportamento instável, se estivermos a falar de intervalos de chegada e de tempos de processamento não determinísticos. No caso de os tempos serem determinísticos, o seu comportamento é instável se houver uma perturbação no sistema, como por exemplo uma avaria num dos servidores.

Os valores para as taxas de chegada e para os tempos médios de processamento são representados na Tabela 5.1 e a verificação da condição de intensidade de tráfego nas equações 5.1.

Tabela 5.1: Parâmetros da rede.

Parâmetro	Valor	Parâmetro	Valor
λ_2	1.0	MPT_5	0.001
λ_3	1.0	MPT_6	0.9
λ_{10}	1.0	MPT_7	0.9
λ_{11}	1.0	MPT_8	0.001
MPT_1	0.9	MPT_9	0.001
MPT_2	0.001	MPT_{10}	0.001
MPT_3	0.001	MPT_{11}	0.001
MPT_4	0.001	MPT_{12}	0.9

$$\begin{aligned}
\rho_1 &= \lambda_1 \times MPT_1 + \lambda_2 \times MPT_2 + \lambda_3 \times MPT_3 = 0.902 \\
\rho_2 &= \lambda_4 \times MPT_4 + \lambda_5 \times MPT_5 + \lambda_6 \times MPT_6 = 0.902 \\
\rho_3 &= \lambda_7 \times MPT_7 + \lambda_8 \times MPT_8 + \lambda_9 \times MPT_9 = 0.902 \\
\rho_4 &= \lambda_{10} \times MPT_{10} + \lambda_{11} \times MPT_{11} + \lambda_{12} \times MPT_{12} = 0.902
\end{aligned} \tag{5.1}$$

Os valores de λ são sempre um também para os buffers internos porque não há routing dinâmico nem merge nem split e a carga imposta aos servidores que os alimentam é sempre inferior a um. É preciso sublinhar que dizer-se que a taxa de chegada aos buffers internos é igual à taxa de chegada aos buffers externos pressupõe que a política de sequenciamento a usar garante estabilidade. Ou seja, que não há perda de capacidade nos servidores por decisões de sequenciamento inadequadas.

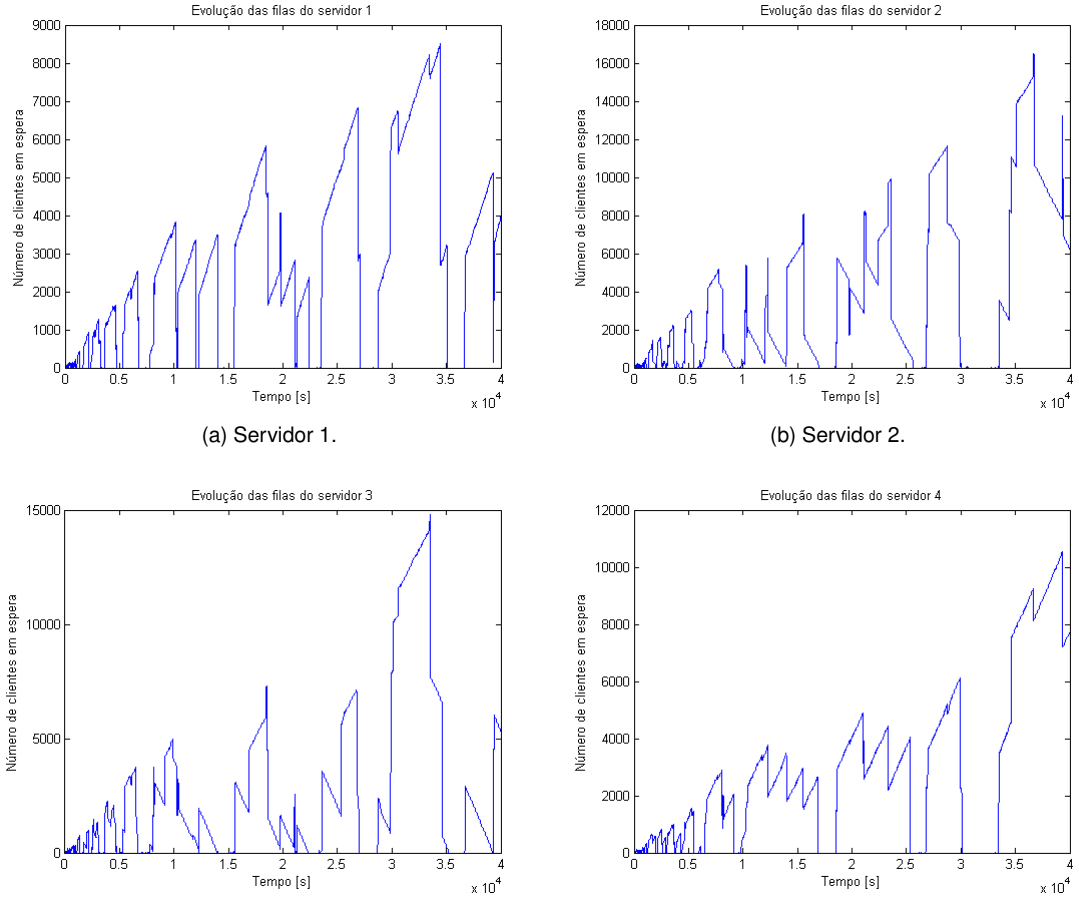
Relativamente ao ficheiro de entrada, além dos campos que caracterizam a rede e são imprescindíveis para a sua criação, os vectores ξ, custos e delta não são precisos e por isso ficam vazios. Para a simulação desta rede utilizou-se sempre o tipo de bloqueio 1 (Equação 4.4), pois cada actividade só é processada por um servidor e a política de sequenciamento também é sempre FIFO relativo.

Utilizando o simulador com o controlador desligado obtiveram-se os resultados da Figura 5.2. Estes resultados foram consistentes com os obtidos pelo José Moreira na sua tese, [4]. Como se pode confirmar, a rede é instável. Por outro lado, Seidman refere em [3] que, partindo de um estado inicial em que não existe ninguém na rede e se não se introduzir nenhuma perturbação, utilizando tempos determinísticos e sem que ocorra alguma avaria num dos quatro servidores, as filas vão-se manter sempre vazias. Estes resultados foram confirmados neste trabalho. Como não possuem relevância especial não são aqui ilustrados.

Os gráficos da Figura 5.2 mostram o número de clientes em espera para cada um dos 4 servidores da

rede. Cada um destes gráficos representa o total de clientes à espera de serem processados por esse servidor que, no caso desta rede, corresponde à soma do tamanho de 3 buffers.

Na Figura 5.3 mostra-se o balanço do número de clientes que entrou no sistema desde o início da simulação, face ao número de clientes que saíram. Este gráfico mostra claramente a evolução da rede. Se a % diminui, significa que a rede é instável, se a % aumenta significa que a rede está a caminhar para a estabilidade. A situação ideal é atingir os 100%, que significa que as filas são finitas sem valor médio. Ou seja, que o valor da espera de alguém que entre no sistema até que sai possui um valor esperado finito.



(a) Servidor 1.

(b) Servidor 2.

(c) Servidor 3.

(d) Servidor 4.

Figura 5.2: Inventário do número de clientes em cada um dos servidores com controlador inactivo.

Na Figura 5.4 mostram-se os resultados obtidos com os mesmos parâmetros de entrada mas com o controlador activado, para um $ajuste = 0$. Ou seja, só se dá o valor da $quota$ (Equação 4.4) sem mais nenhum 'extra'. A Figura 5.5 representa o respectivo balanço de entradas e saídas.

Como se pode ver pela escala das figuras 5.4, a rede ficou estável. Embora existam algumas flutuações, não há comparação possível com os resultados obtidos sem controlador. Sobrepondo os dois resultados obteve-se o ilustrado nas figuras 5.6 e 5.7.

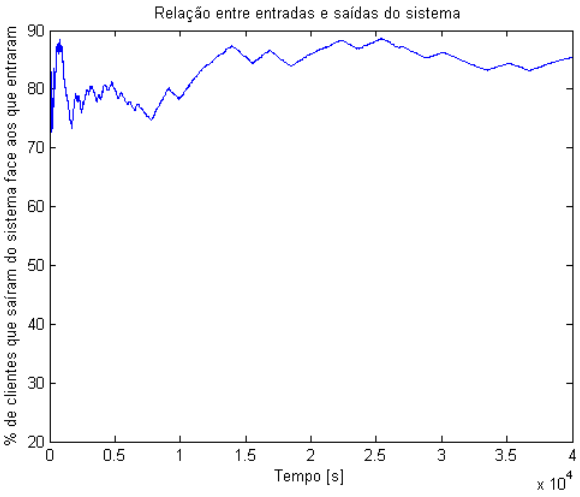
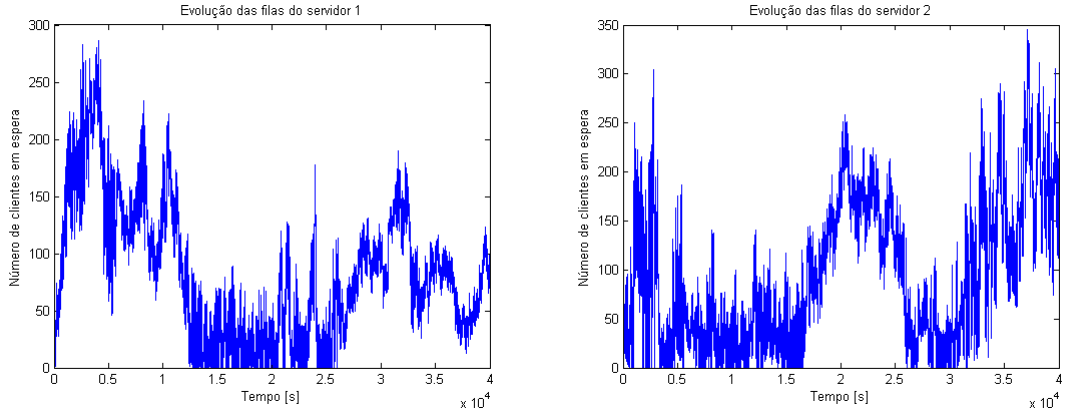
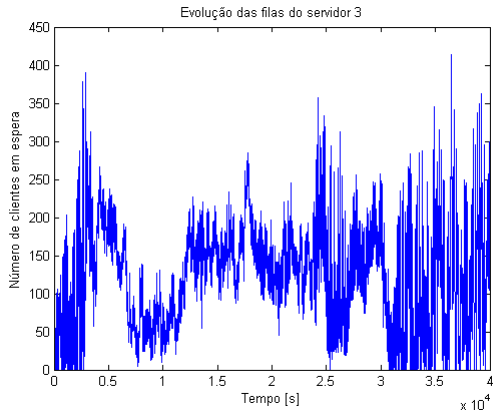


Figura 5.3: Balanço com controlador inactivo.

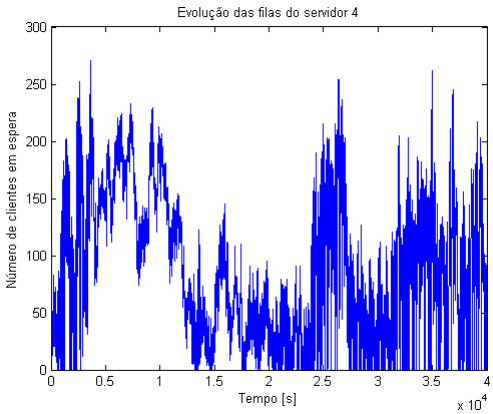


(a) Servidor 1.

(b) Servidor 2.



(c) Servidor 3.



(d) Servidor 4.

Figura 5.4: Inventário de clientes em espera com controlador e sem $ajuste$ em θ.

De seguida mostram-se os resultados obtidos por várias simulações com vários valores para o $ajuste$ de θ^i_j. Testaram-se valores de $ajuste$ que variam entre valores negativos e positivos. Só deste modo se pode observar a alteração de comportamento da rede quando se dá, ou se tira, mais $quota$ do que a

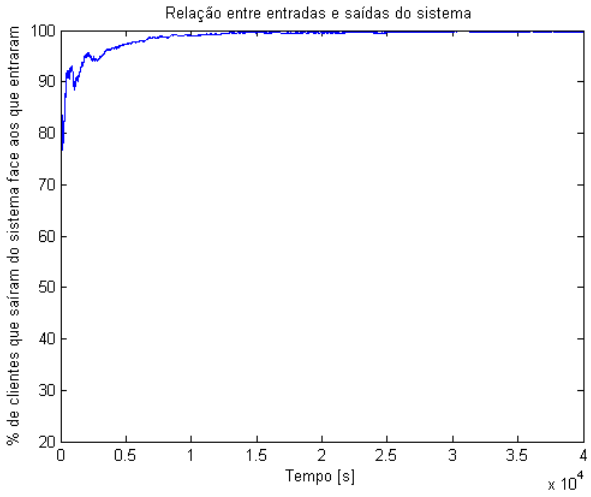
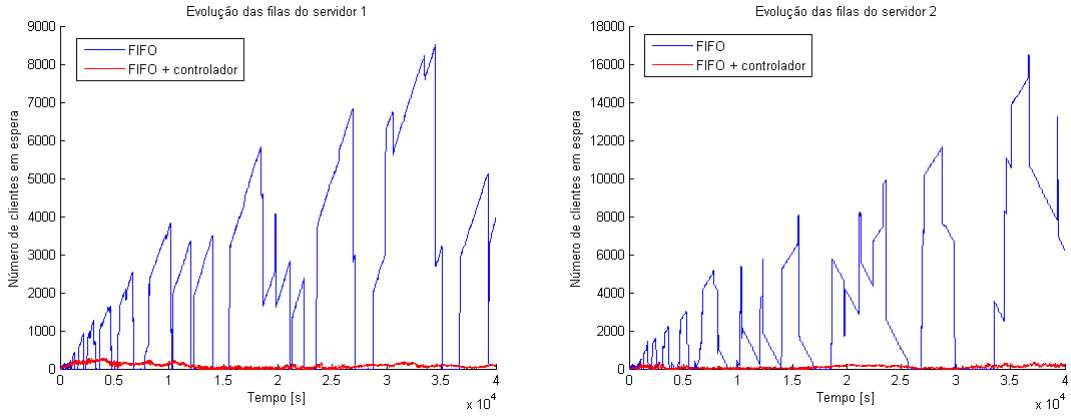
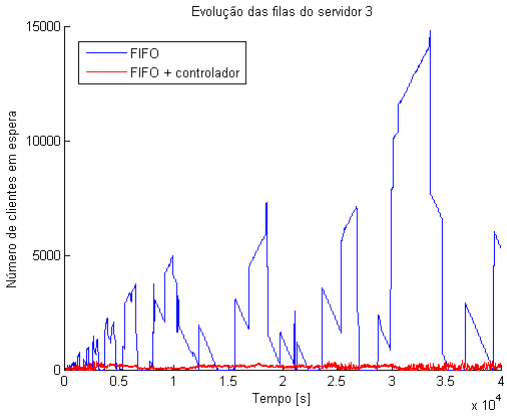


Figura 5.5: Balanço com controlador e sem $ajuste$ em θ.

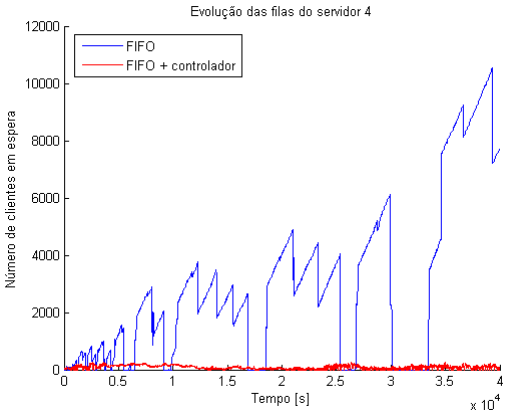


(a) Servidor 1.

(b) Servidor 2.



(c) Servidor 3.



(d) Servidor 4.

Figura 5.6: Comparação do inventário de clientes em espera com e sem controlador.

estritamente necessária. Vai-se então começar por analisar os casos em que $k < 0$.

As figuras 5.8 e 5.9 foram obtidas para um valor de $k = -500000$. Ou seja, é retirada $quota$. No entanto, θ_j^i é um valor é muito próximo de zero.

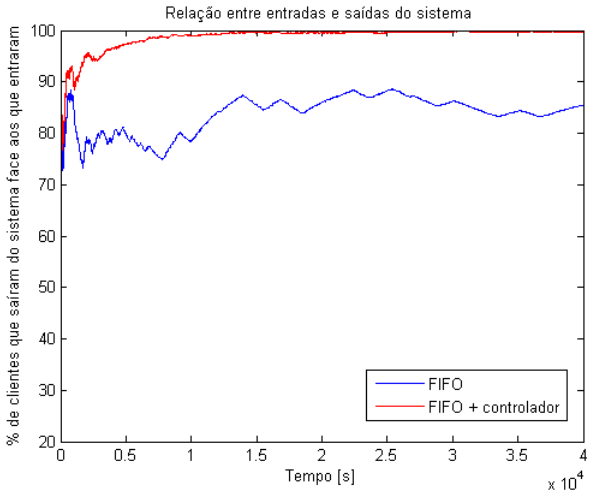
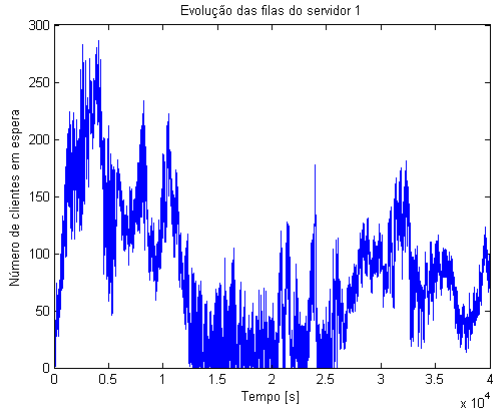
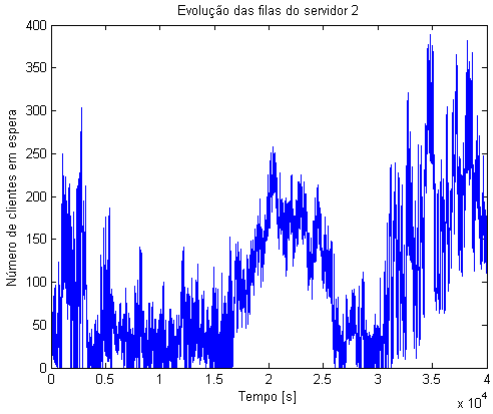


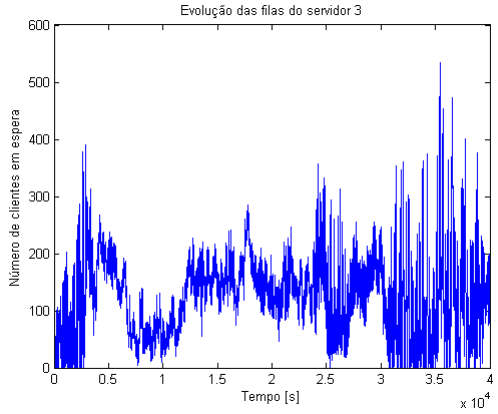
Figura 5.7: Comparação dos balanços com e sem controlador.



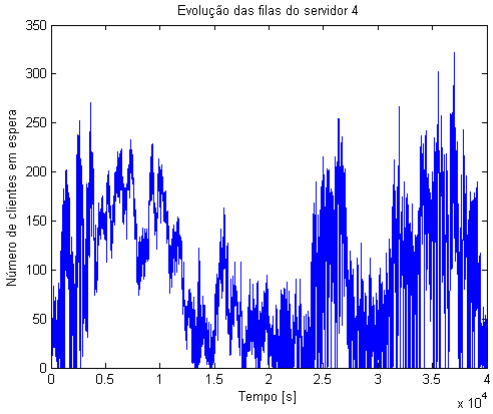
(a) Servidor 1.



(b) Servidor 2.



(c) Servidor 3.



(d) Servidor 4.

Figura 5.8: Inventário do número de clientes em cada um dos servidores com controlador activo e $k = -500000$.

No caso das figuras 5.10 e 5.11 o valor de k já é 5 vezes maior do que no das figuras 5.8 e 5.9 ($k = -100000$). No entanto, o seu módulo é menor, e portanto, o módulo do ajuste é maior, ou seja, é retirada ainda mais capacidade. Mesmo que o sistema se consiga manter inicialmente estável, uma

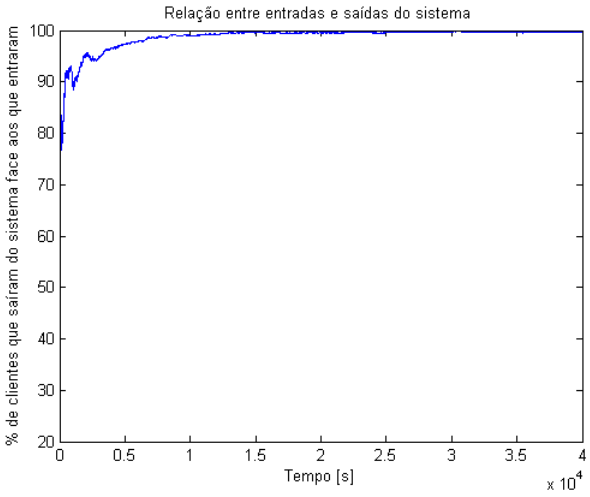
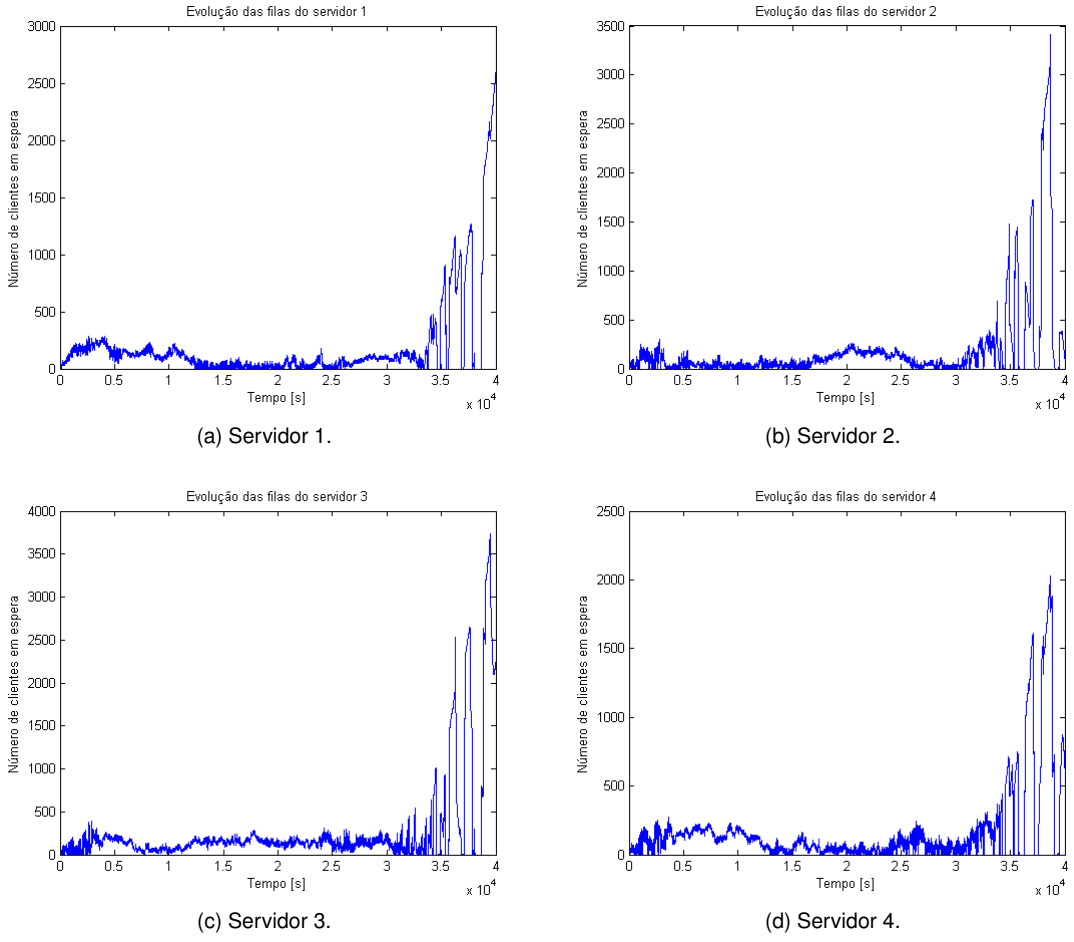


Figura 5.9: Balanço com controlador e $k = -500000$.



(a) Servidor 1.

(b) Servidor 2.

(c) Servidor 3.

(d) Servidor 4.

Figura 5.10: Inventário do número de clientes em cada um dos servidores com controlador activo e $k = -100000$.

vez que entre em desequilíbrio nunca mais volta a ganhar estabilidade.

Nas figuras 5.12 e 5.13 simulou-se para um valor de $k = -1000$. Pode-se verificar que tal como para o caso de $k = -100000$ o sistema ao fim de pouco tempo fica instável. Neste caso demora menos tempo

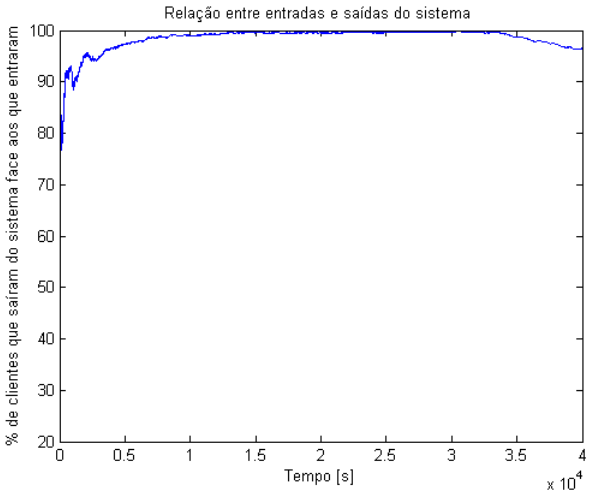
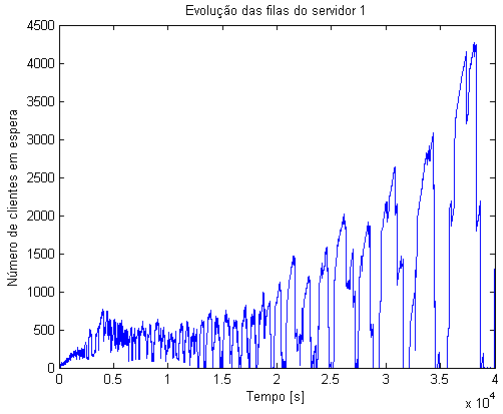
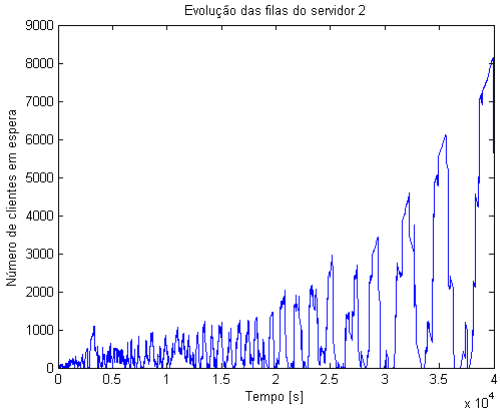


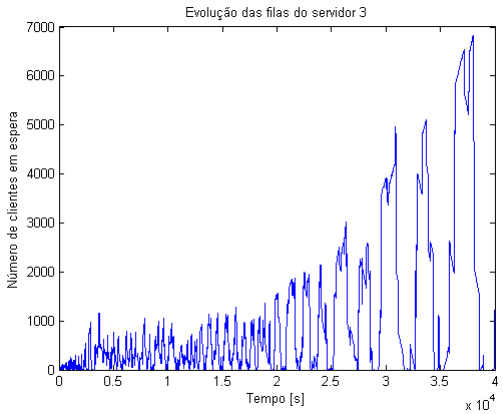
Figura 5.11: Balanço com controlador e $k = -100000$.



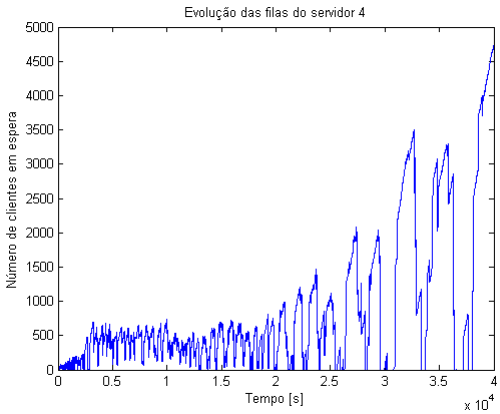
(a) Servidor 1.



(b) Servidor 2.



(c) Servidor 3.



(d) Servidor 4.

Figura 5.12: Inventário do número de clientes em cada um dos servidores com controlador activo e $k = -1000$.

a perder a estabilidade porque o $|k|$ é menor, o que significa que se retirou mais quota que na simulação em que $k = -100000$. O único caso aqui apresentado aparentemente estável é o de $k = -500000$, no entanto, se se observasse apenas metade da simulação para $k = -100000$ provavelmente também se

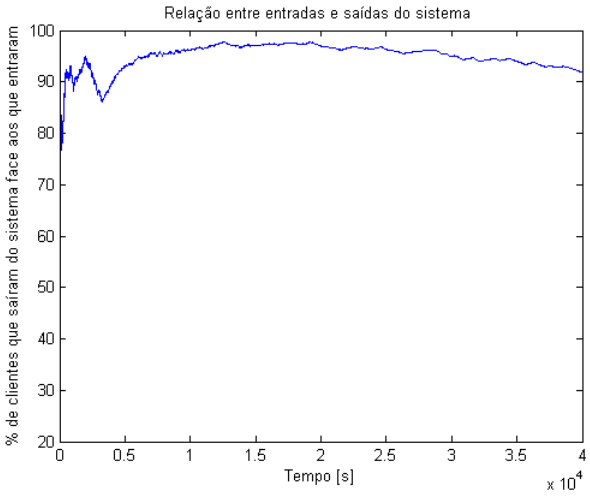


Figura 5.13: Balanço com controlador e $k = -1000$.

concluiria que o sistema estava estável. O que leva a induzir que se se corresse a simulação durante mais tempo também o caso de $k = -500000$ viria a perder estabilidade nalgum ponto.

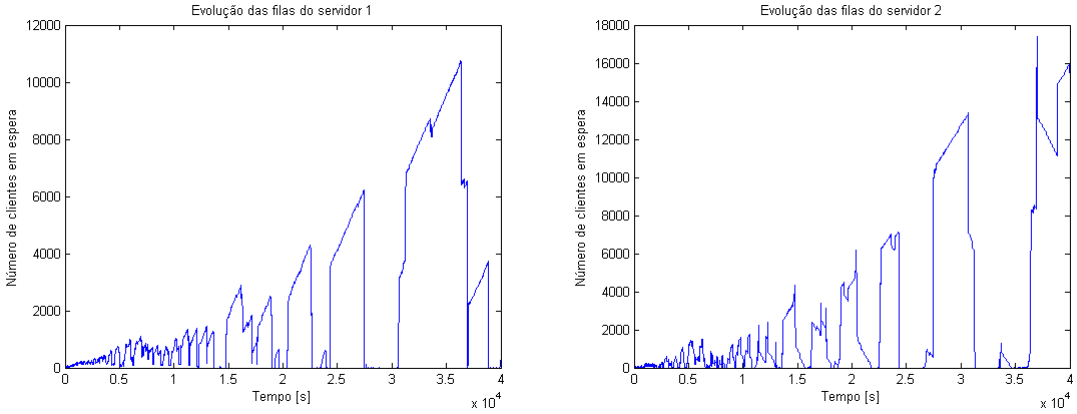
Em seguida vão-se analisar os resultados obtidos quando $k > 0$. Nestes casos como k é positivo o $ajuste$ é positivo, o que corresponde a dar um acréscimo à $quota$ pré-definida. Como se pode ver pelas figuras 5.14 e 5.15, a rede é instável. Significa que a $quota$ que se está a dar a cada um dos buffers é excessiva, o que corresponde quase à situação sem controlador (em que não há $quota$ definida).

Aumentando o valor de k, de 10000 para 100000, diminui-se o valor do $ajuste$ adicionado à $quota$. O valor deste aproxima-se de zero, e é por isso mais limitativo que o caso anterior. O tempo de bloqueio aumenta, porque a condição 4.1 deixa de se verificar com mais frequência e a rede mantém-se estável.

Tornando a aumentar o valor de k desta vez para 500000 o comportamento mantém-se. Mais uma vez reduzindo-se o valor do $ajuste$ adicionado à $quota$, o tempo de bloqueio manteve-se na mesma ordem de grandeza. Sofreu apenas um aumento muito pequeno, mas foi o suficiente para que o desempenho melhorasse. O valor do pico atingido no servidor 2 passou de um máximo de aproximadamente 500 clientes em espera para 350 e o servidor 3 de 700 para 400 clientes.

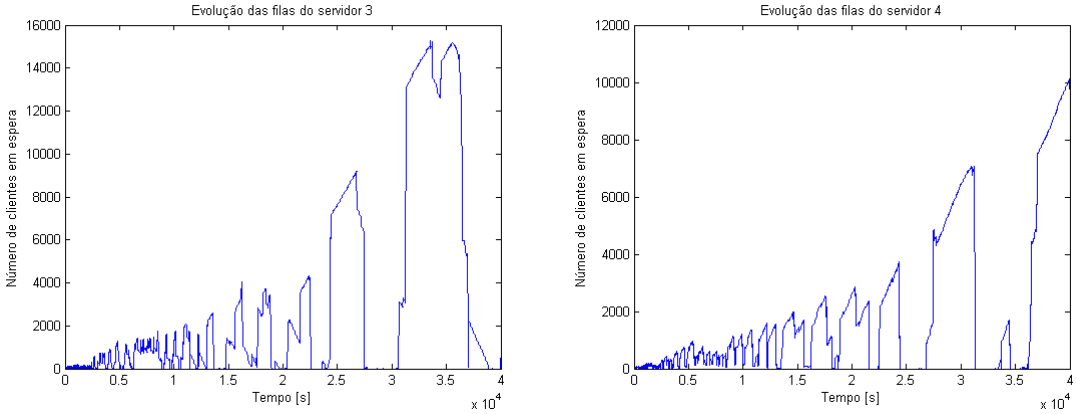
Os resultados obtidos para valores de $k > 500000$ já não mostram nenhuma melhoria significativa. Mesmo este caso é bastante comparável ao primeiro aqui apresentado em que o valor do $ajuste$ é igual a zero.

Já se fez uma análise geral do comportamento da rede para diferentes valores de k, no entanto a relação entre estes desempenhos e os respectivos tempos de bloqueio dos buffers ainda não foi vista em detalhe. Analisando a tabela 5.2 pode-se verificar o seguinte: para valores de $k > 0$, quando maior for o seu módulo, menor é o ϵ e menos $quota$ 'extra' é dada a cada um dos buffers. Este fenómeno faz com que os buffers sejam bloqueados com mais frequência, porque a condição 4.1 é violada mais



(a) Servidor 1.

(b) Servidor 2.



(c) Servidor 3.

(d) Servidor 4.

Figura 5.14: Inventário do número de clientes em cada um dos servidores com controlador activo e $k = 10000$.

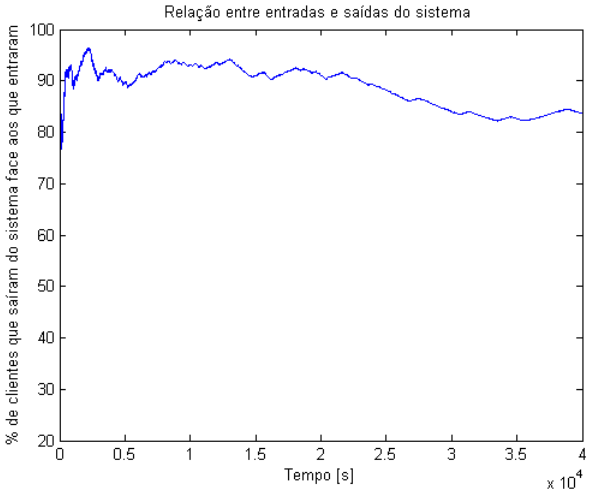
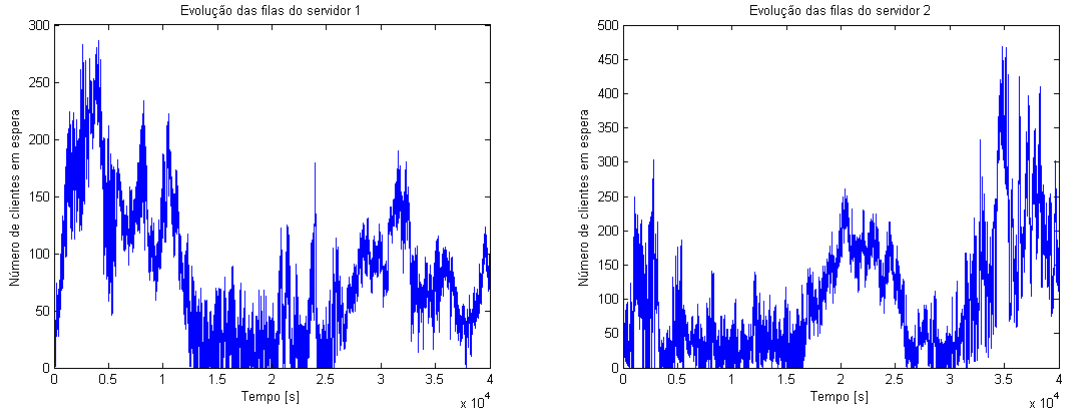


Figura 5.15: Balanço com controlador e $k = 10000$.

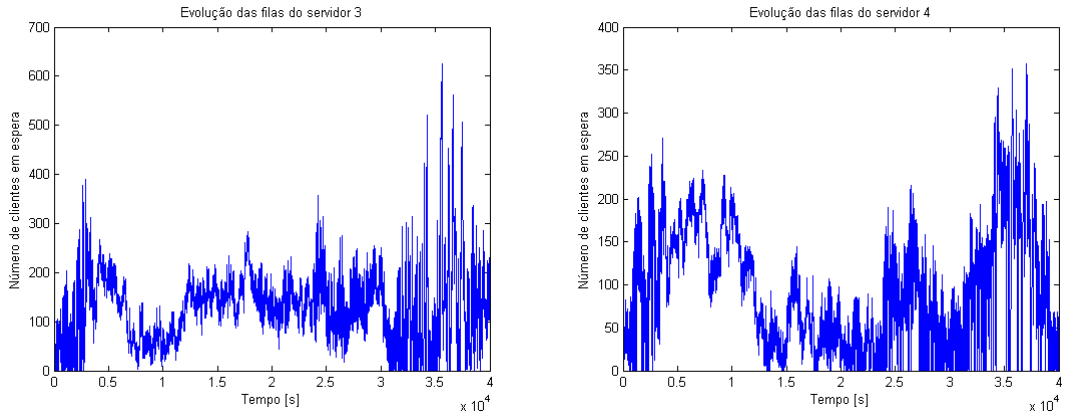
facilmente e o tempo de bloqueio é maior. O caso em que não existe controlador não é mais do que um valor de $\theta_j^i = \infty$. Ou seja, a condição 4.1 nunca é violada e nunca nenhum buffer é bloqueado.

Se estivermos a falar de valores de $k < 0$ o comportamento já não é tão linear. Quanto maior for o seu



(a) Servidor 1.

(b) Servidor 2.



(c) Servidor 3.

(d) Servidor 4.

Figura 5.16: Inventário do número de clientes em cada um dos servidores com controlador activo e $k = 100000$.

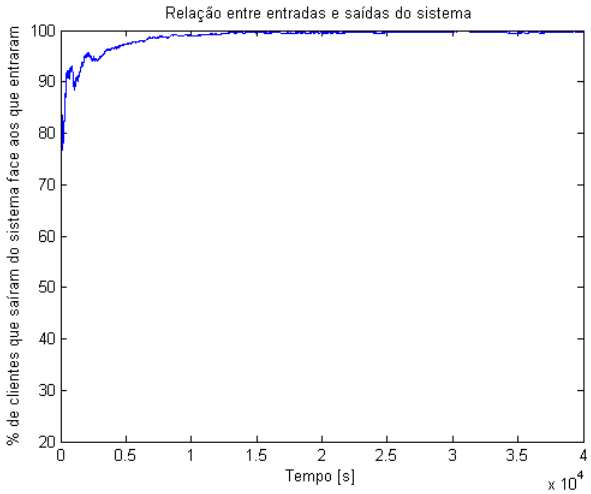
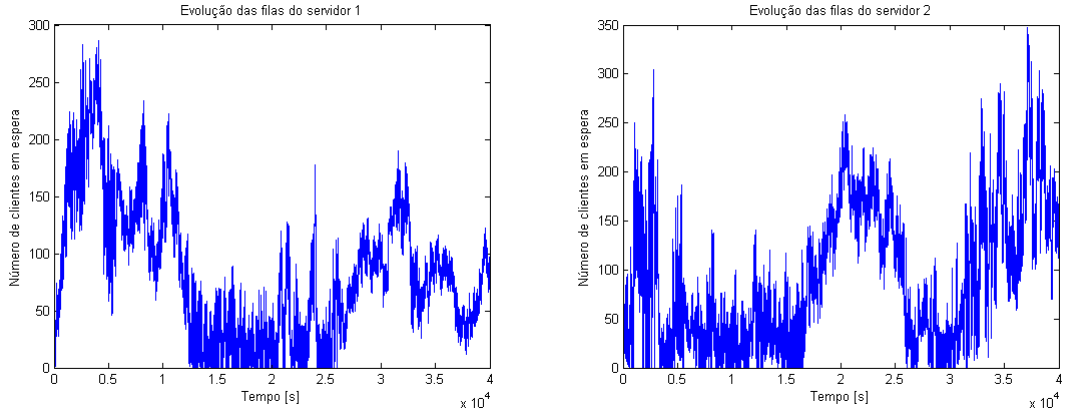


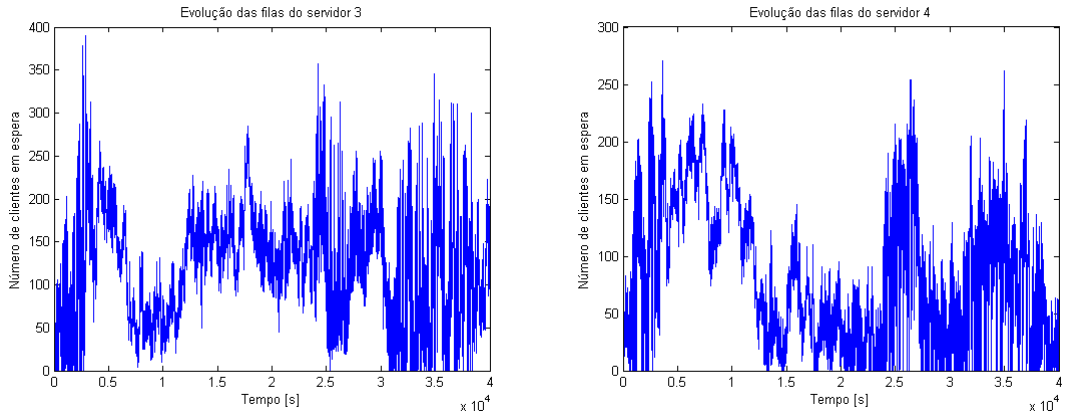
Figura 5.17: Balanço com controlador e $k = 100000$.

módulo, menos $quota$ é retirada, e mais próximo fica θ_j^i da $quota$ pré-definida. Por outro lado, quanto menor for o módulo, mais longe fica θ_j^i da quota pré-definida. O sistema entra em desequilíbrio mais rapidamente porque θ_j^i é pequeno demais e mais cedo os buffers começam a ser bloqueados. No



(a) Servidor 1.

(b) Servidor 2.



(c) Servidor 3.

(d) Servidor 4.

Figura 5.18: Inventário do número de clientes em cada um dos servidores com controlador activo e $k = 500000$.

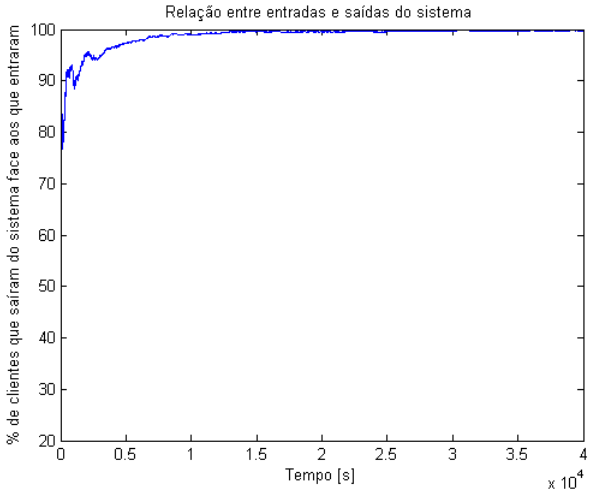


Figura 5.19: Balanço com controlador e $k = 500000$.

entanto, como o sistema perde estabilidade, muito mais dificilmente torna a atingir os limites, e por isso o tempo de bloqueio no final da simulação é muito menor.

Finalmente, a Tabela 5.3 mostra os tempos de inactividade passiva e activa relativamente a cada uma

Tabela 5.2: Tempos de bloqueio dos buffers.

ID	Valor de k							
	0	∞	-500000	-100000	-1000	10000	100000	500000
1	0	1500	1500	829	0	0	1500	1500
2	0	2271	2272	2272	2108	2498	2273	2271
3	0	4943	4947	4910	2624	2586	4911	4942
4	0	11949	11534	8304	517	331	11474	11966
5	0	883	847	886	76	69	879	882
6	0	249	181	180	0	13	235	248
7	0	1.44	1.44	1.44	4.96	1.43	1.44	1.44
8	0	15924	15906	15843	1798	1518	16016	15961
9	0	6997	6856	6555	107	121	7115	7601
10	0	6806	6797	6888	3434	3356	6844	6815
11	0	4778	4770	4869	1669	2249	4878	4827
12	0	679	679	684	0	0	684	679

das simulações referidas. É importante observar que cada um dos servidores tem uma folga ligeiramente inferior a 10%. Por isso, tendo as simulações a duração de 40000 unidades de tempo, o tempo de inactividade em cada um dos servidores deve rondar as 4000 unidades de tempo. Se exceder este valor de forma significativa, a rede fica instável. Tal como se pode verificar pelos resultados apresentados, nos casos em que a rede é instável esse valor é sempre ultrapassado de forma substancial (ver $k = -1000$ em que a inactividade total varia entre os 16% e os 18.5%). Nos casos em que a rede é estável, o total de inactividade anda sempre perto de 10% (ver para $k = 100000$ em que a inactividade total varia entre os 9.79% e 10.7%). Para os casos em que a rede é estável, apesar de aquele valor ser ultrapassado em algumas situações, importa sublinhar que, tratando-se de simulação estocástica e estando a observar um só percurso amostra o valor 4000 é meramente indicativo. Admitem-se ligeiras flutuações em torno daquele valor para estabilidade, mas a instabilidade, quando ocorre, produz desvios muito significativos.

5.2 Estudo da rede de Lu *et al.*

A secção anterior tinha como objectivo mostrar que a inactividade pode ser benéfica para estabilização de redes de filas de espera. No entanto, existem políticas para a qual a rede é estável. Se assim é, então porque é que alguém se dá ao trabalho de arranjar um mecanismo de estabilização de uma rede que com outra política já é estável? A resposta é simples. Pode ser que uma política que faça com que a rede seja instável, com o controlador não só fique estável, como tenha melhor desempenho do que uma política que já à partida seja estável.

Melhorar desempenhos foi então o objectivo no estudo da rede de Lu *et al.*, [1]. A rede aqui apresentada é mais uma vez uma rede de filas de espera. Cada actividade tem um buffer de entrada e um outro de saída. Neste caso existem servidores 'gémeos', ou seja, há actividades que são processadas por

Tabela 5.3: Tempos de inactividade dos servidores.

	Tipo de Inactividade			Tipo de Inactividade	
	Passiva	Activa		Passiva	Activa
$k = 0$:			$k = -1000$:		
Servidor 1	9052.958	0	Servidor 1	5351.413	1624.212
Servidor 2	9084.354	0	Servidor 2	6231.802	209.545
Servidor 3	9386.235	0	Servidor 3	6683.744	749.290
Servidor 4	9272.883	0	Servidor 4	4413.457	2239.989
$k = \infty$:			$k = 10000$:		
Servidor 1	1218.53	2698.234	Servidor 1	7663.295	1996.098
Servidor 2	1416.687	2643.023	Servidor 2	9524.129	173.184
Servidor 3	1227.261	3064.833	Servidor 3	9117.171	874.842
Servidor 4	2372.009	1851.198	Servidor 4	7041.878	2896.804
$k = -500000$:			$k = 100000$:		
Servidor 1	1218.655	2698.120	Servidor 1	1222.647	2694.076
Servidor 2	1460.038	2569.033	Servidor 2	1423.253	2612.900
Servidor 3	1272.376	2993.093	Servidor 3	1308.959	2969.184
Servidor 4	2330.425	1852.409	Servidor 4	2297.471	1899.370
$k = -100000$:			$k = 500000$:		
Servidor 1	2939.191	2044.701	Servidor 1	1217.749	2699.008
Servidor 2	3050.729	2165.364	Servidor 2	1416.969	2611.918
Servidor 3	2631.300	2843.245	Servidor 3	1077.228	3187.339
Servidor 4	3404.966	1954.809	Servidor 4	2342.158	1843.239

mais do que um servidor. A topologia da rede é a da Figura 5.20. Como se pode ver cada cliente faz na maioria dos casos mais do que uma visita a cada uma das estações da fábrica. Os parâmetros da rede estão representados na Tabela 5.4.

A taxa de chegada ao sistema é de 0.0236 e as condições da simulação foram semelhantes às usadas por Lu *et al.*. Todas as simulações foram feitas utilizando como critério de paragem o número de clientes que saiu do sistema. Simulou-se sempre com a rede inicialmente vazia para um total de 3000 clientes em que apenas os últimos 1000 foram considerados para cálculo de medidas de desempenho.

Lu et al. mostraram que uma subclasse das políticas de *least slack*, denominadas *Fluctuation Smoothing Policies* (FS) são as que produzem melhores resultados a nível de redução do tempo médio de ciclo (FSMCT) e da variância do tempo de ciclo (FSVCT).

Para estas políticas é preciso ter acesso a uma estimativa dos tempos de ciclo restantes para todos os buffers, ou seja, do vector ξ. Primeiro é preciso encontrar uma estimativa destes valores para poder simular. Utilizando o mesmo método de convergência de Lu *et al.*, começa-se por inicializar o vector a zero, $\xi_i = 0$ para todos o i. De seguida simula-se com o vector ξ_i e obtém-se uma estimativa para o vector $\hat{\xi}$ que, por sua vez, vai passar a ser o vector de entrada $\xi_i^{(1)} = \hat{\xi}_i^{(0)}$. Este processo repete-se sucessivamente, mas como não há propriamente um processo de convergência basta fazerem-se 10 iterações e escolhe-se para vector aquele que produz um melhor resultado.

Como esta rede só tem um tipo de clientes e só há um caminho possível, basta comparar ξ_1, que é o tempo restante de ciclo para um cliente que se encontre no buffer de entrada do sistema. Para

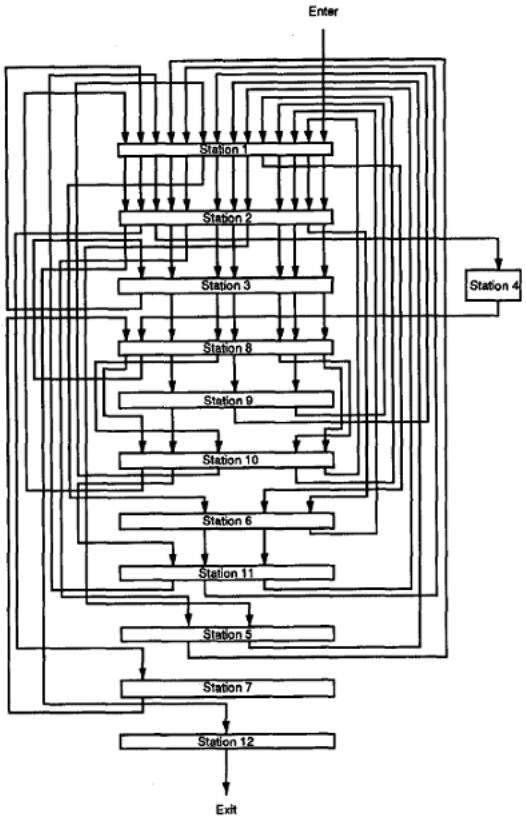


Figura 5.20: Planta da fábrica de manufactura de semiconductores.

todos os outros buffers o valor vai sendo progressivamente mais pequeno, uma vez que o cliente se vai aproximando da saída da rede. Por isso, se o valor do tempo de ciclo restante no primeiro buffer baixa, os outros também baixam todos.

Como se pode ver pela Tabela 5.5, não há convergÊncia uma vez que os resultados sucessivos não exibem comportamento monotónico. Tendo em conta os resultados tentou-se descobrir um método que convergisse. Começou-se por simular com a condição inicial $\xi_i^{(0)} = 0$ para todos os i e, em vez de utilizar a estimativa obtida para nova simulação, utilizaram-se percentagens dessa estimativa para fazer as simulações. Ou seja, o vector de entrada de cada estimativa obtida na Tabela 5.6 corresponde a uma percentagem do $\xi_i^{(1)} = \hat{\xi_i}^{(0)}$. Essa percentagem vai desde 10% até 200% e aumenta de 10% em 10%.

Como se pode ver pelos resultados obtidos nesta tabela, este método também não é eficaz. Nem se conseguiu arranjar outro melhor, devido à complexidade da política em si, e ao número de buffers associados, que é bastante elevado (172). No entanto, fica feita a ressalva de que deve haver um método mais eficiente para a convergência dos tempos restantes de ciclo.

O próximo passo é então simular o comportamento da rede com e sem controlador. Como já foi referido, nesta rede existem actividades que são processadas por mais do que um servidor. Por isso, o tipo de bloqueio especificado no ficheiro de entrada é o 2. Neste caso, o vector δ tem de vir preenchido com

Tabela 5.4: Parâmetros da rede.

Station	Num máquinas (Q)	Num visitas	MPT
1	2	19	1.55
2	2	5	4.98
3	2	5	5.45
4	1	3	4.68
5	1	1	6.14
6	1	2	7.76
7	1	1	6.23
8	1	3	4.35
9	1	2	4.71
10	1	3	4.05
11	1	1	7.86
12	1	2	6.10
13	4	13	4.23
14	3	12	7.82
15	1	15	0.87
16	2	11	2.96
17	1	10	1.56
18	1	4	3.59
19	2	2	13.88
20	1	2	5.41
21	2	4	7.58
22	2	21	1.04
23	2	23	1.09
24	2	8	3.86

Tabela 5.5: Convergência do ξ.

Iteração	ξ_1	TMPS
0	712.856	713.291
1	707.867	708.144
2	706.903	707.197
3	706.208	706.492
4	706.208	706.512
5	708.005	708.377
6	706.225	706.536
7	706.640	706.916
8	707.032	707.336
9	703.709	704.102
10	707.204	707.5492

Tabela 5.6: Convergência do ξ suavizada.

Iteração	ξ_1	TMPS	Iteração	ξ_1	TMPS
1	712.856	713.291	11	708.215	708.451
2	713.812	714.176	12	708.746	708.980
3	709.335	709.802	13	712.682	712.926
4	710.371	710.796	14	711.005	711.235
5	711.230	711.582	15	713.022	713.268
6	711.557	711.860	16	714.255	714.532
7	712.206	712.523	17	716.209	716.368
8	709.814	710.127	18	727.645	727.847
9	706.257	706.583	19	721.221	721.361
10	707.867	708.144	20	729.442	729.598

os valores para cada um dos buffers. Como o que que se pretende é regular as admissões ao sistema, basta activar o bloqueio para o primeiro buffer, ou seja, apenas a primeira posição do vector é que é diferente de zero.

Repare-se que se $\delta = 0$ o buffer respectivo nunca bloqueia, porque neste caso a condição de bloqueio passa a ser a da representada na Equação 5.2.

$$t_{dedicado} > t_{actual} * q_j \tag{5.2}$$

Condição essa impossível de acontecer, uma vez que o tempo dedicado por todos os servidores a uma actividade nunca pode ser superior à duração da simulação vezes o número de servidores que a processam. Essa situação significava que os servidores dedicavam a essa actividade mais tempo do que o tempo decorrido desde o início da simulação.

Na tabela 5.7 apresentam-se os resultados obtidos sem controlador (caso em que $\delta = 0$) e com vários valores de δ, utilizando a política FSMCT. Repetiu-se o mesmo estudo para a política LBFS. Mais uma vez basta fazer a análise para o buffer de entrada.

Tabela 5.7: Resultados obtidos com controlador para fábrica ([1]) com política FSMCT.

δ_1	ξ_1	TMPS	Tempo Bloqueio
0	703.709	704.102	0
1.9611	703.709	704.102	2.557
1.9612	703.709	704.102	5.940
1.9613	703.709	704.102	9.883
1.9614	703.709	704.102	13.969
1.9615	703.709	704.102	18.077
1.9616	703.709	704.102	23.982
1.9617	703.709	704.102	34.608
1.9618	703.709	704.102	48.373
1.9619	703.709	704.102	59.585
1.96195	703.709	704.102	64.982
1.96198	703.709	704.102	68.286
1.961985	703.709	704.102	68.838
1.961986	707.568	707.902	68.948
1.9619875	707.157	707.452	69.113
1.96199	705.068	705.362	69.389
1.962	707.568	707.902	70.493
1.9625	711.269	711.616	134.806
1.965	751.537	2339.47	70309.022
1.97	687.766	20062.7	137011.824

Analisando os resultados para a política FSMCT verifica-se que embora os resultados não piorem, também não melhoram. Já para a política LBFS conseguem-se resultados melhores. No entanto, mesmo assim, o desempenho continua a ser melhor com a política de *least slack* com ou sem controlador.

Como o objectivo é conseguir melhores resultados que a melhor política que actualmente se consegue

Tabela 5.8: Resultados obtidos com controlador para fábrica ([1]) com política LBFS.

δ	ξ_1	TMPS	Tempo Bloqueio
0	713.924	714.258	0
1.9615	713.924	714.259	17.162
1.962	713.924	714.258	70.600
1.963	713.924	714.258	304.811
1.9631	713.924	714.258	345.332
1.9632	712.724	713.154	365.174
1.9633	713.259	713.629	244.354
1.9634	762.952	774.979	356.049
1.9635	712.488	712.912	278.545
1.96355	763.686	776.797	436.028
1.964	712.724	713.154	279.106
1.9645	755.432	1037.15	21213.627
1.965	767.242	2358.85	67232.882
1.97	690.242	2358.850	130997.388

arranjar, facilmente se conclui que este objectivo não foi bem sucedido.

No caso desta tese, o tamanho da janela do controlador, é do mesmo tamanho da simulação. Comparando com os resultados de José Moreira, que tinha um controlador com o tamanho da janela limitado, facilmente se conclui que a solução ideal talvez seja uma solução intermédia destas duas. Essa solução vai ser apresentada no capítulo das conclusões e trabalho futuro.

Capítulo 6

Conclusões e Trabalho Futuro

6.1 Conclusões

Após a análise dos resultados obtidos as principais conclusões a tirar são as seguintes:

- Inactividade activa não é um desperdício de recursos e pode ser uma ferramenta para estabilizar redes com servidores que efectuem mais do que uma actividade;
- O controlador implementado é uma realização possível que introduz este tipo de inactividade. No entanto, embora sirva para estabilização, aparenta não ser suficiente para melhoria de desempenho em redes cujas políticas asseguram estabilidade.

O primeiro item representa o resultado mais relevante deste trabalho. Apesar de existirem, para o caso estudado nesta tese, políticas non-idling que garantem a estabilidade da rede, mostrou-se que a introdução de inactividade em alguns momentos pode ser benéfica para o desempenho da mesma. A rede de Seidman que perante a política de sequenciamento FIFO relativo era instável, tornou-se estável pelo simples facto de impor quotas a cada um dos buffers ao longo do tempo. Tal como já foi referido, a imposição de quotas faz com que, independentemente das prioridades, que não são de modo algum alteradas, não se deixe que um servidor dedique demasiado tempo a um buffer sob pena de deixar de processar outros durante esse intervalo.

Não alterando as prioridades dos buffers o que acontece é que quando a quota de um buffer é ultrapassada, esse buffer torna-se durante um intervalo de tempo invisível ao servidor, o que faz com que o servidor comece a processar clientes de outros buffers. O objectivo não é nunca que o servidor fique inactivo, mas sim que os buffers de saída das actividades que este processa não fiquem vazios por muito tempo, de modo a que os restantes servidores da rede não fiquem inactivos por não terem

clientes para processar.

Confirmou-se que se se $\theta_j^i < quota$, mais cedo ou mais tarde a rede se torna instável. Por outro lado, também se verificou que se $\theta_j^i \ggg quota$, o comportamento da rede se aproxima da situação em que não existe controlador.

Este tipo de controlador só faz sentido nos servidores que processam mais do que uma actividade, ou seja, servidores multiclasse. Uma vez que para um servidor uniclasse e com routing estático não há nenhum tipo de vantagem em deixar atrasar trabalho.

Abre-se assim um novo horizonte a nível de políticas de sequenciamento. Políticas idling podem efectivamente ser melhores que políticas non-idling.

Relativamente a melhorar desempenhos, este objectivo já não foi atingido com o sucesso esperado.

Como pôde ser constatado, existem políticas non-idling cujo desempenho melhora com o controlador implementado. O caso da rede de Lu *et al.* foi um em que de facto se constatou que a rede, quando a política utilizada foi LBFS, tinha um melhor desempenho com controlador do que sem ele. No entanto, este desempenho melhorado ficou aquém do obtido com a política FSMCT.

No caso da política FSMCT não se conseguiu que o seu desempenho melhorasse com controlador. Uma vez que se mostrou em [1] que esta política é a que produz melhores resultados na redução do tempo médio de ciclo, era a que interessava verdadeiramente que fosse melhorada. No entanto, o facto de não se ter conseguido melhorar o desempenho leva-nos a tirar algumas conclusões interessantes.

O tamanho da janela do controlador utilizado talvez seja grande demais. Se a simulação for muito comprida, e até um determinado ponto o tempo dedicado a um determinado buffer for muito pequeno, se a dada altura houver um burst de chegadas a esse buffer, o que acontece é que existe uma folga muito grande até que o buffer seja bloqueado, e por isso esse burst vai ser propagado pela rede, podendo fazer com que a mesma entre em desequilíbrio e se torne instável.

A grande contribuição desta tese foi o simulador de actividades. Há sempre ideias novas de topologias de redes para estudar e não faz sentido ter de criar métodos de simulação para cada uma delas. Deste modo basta apenas seleccionar o ficheiro de entrada e os respectivos parâmetros. Os ficheiros de saída fornecem todos os dados necessários para qualquer cálculo.

6.2 Trabalho Futuro

Há várias sugestões de desenvolvimento futuro para esta tese. Em primeiro lugar, e por uma questão de versatilidade do simulador, fica a sugestão de implementar mais métodos para o processo de chegada

às redes além da actualmente disponível, e adicionar essa opção à interface.

Actualmente, o ficheiro de entrada precisa de ter todas as linhas especificadas, mesmo que estas não sejam importantes para a simulação em questão. A sugestão que aqui se deixa é a de tornar a leitura do ficheiro um pouco mais flexível.

Relativamente ao controlador propriamente dito, tal como foi dito na secção anterior, a janela precisa de ser mais bem definida, ficam aqui duas sugestões:

- A primeira está relacionada com o facto de que com o controlador actual, o passado recente pesa tanto como o passado mais antigo. Se o passado recente tiver um peso superior ao do passado mais antigo, um serviço mais comprido que o normal ou um burst de chegadas a uma rede, vai fazer com que o buffer fique bloqueado com mais facilidade. O que se propõe é que, mantendo o tamanho da janela tal como está, se faça uma suavização exponencial de modo a que os tempos dedicados a cada um dos buffers tenham um peso diferente ao longo do tempo. Definindo como $f_j^i(t_k)$ o valor do tempo dedicado pelo servidor j à actividade i até ao instante t_k depois de suavizado exponencialmente a actualização far-se-à tendo em conta os seguintes casos:

  Caso A: Início de serviço. Seja t_{k-1} o fim do último serviço nesta actividade por este servidor e t_k o início do próximo serviço, então:

  $$f_j^i(t_k) = f_j^i(t_{k-1}) \times e^{-\beta(t_{k-1}-t_k)} + \int_{t_{k-1}}^{t_k} \emptyset \times e^{-\beta(t_k-t)}dt \tag{6.1}$$

  Caso B: Fim de serviço. Seja t_{k-1} o início do novo serviço e t_k o fim do serviço então:

  $$f_j^i(t_k) = f_j^i(t_{k-1}) \times e^{-\beta(t_{k-1}-t_k)} + \int_{t_{k-1}}^{t_k} 1 \times e^{-\beta(t_k-t)}dt \tag{6.2}$$

- Em alternativa, uma vez que os buffers são analisados sempre um a um e nunca no contexto da rede em geral, fazer com que no instante em que um buffer é desbloqueado, a sua janela volte ao tamanho inicial, ou seja, zero. Em termos práticos o que isto significa é que de cada vez que um buffer é desbloqueado o contador do tempo dedicado a esse buffer vem a zero. Deste modo, em vez de se comparar o tempo dedicado a cada buffer desde o início da simulação com o tempo da simulação em que esta verificação ocorre, compara-se o tempo dedicado desde que o buffer foi desbloqueado com o tempo que passou desde que esse evento ocorreu.

  $$t_{dedicado\_desde\_ultimo\_reset} > (t_{actual} - t_{ultimo\_reset}) \times \theta \tag{6.3}$$

Bibliografia

[1] Steve C. H. Lu, Deepa Ramaswamy, and P. R. Kumar. Efficent scheduling policies to reduce mean and variance of cycle-time in semiconductor manufacturing plants. *IEEE Transactions Semiconductor Manufacturing*, 7(3):374–388, August 1994.

[2] J. Michael Harrison. Stochastic networks and activity analysis. In *Analytic Methods in Applied Probability*, pages 53–76. American Mathematical Society, 2002.

[3] Thomas I. Seidman. 'first come, first served' can be unstable! *IEEE Transactions on Automatic Control*, 39:2166–2171, 1994.

[4] J. Moreira. Distributed scheduling with active idleness: A key to the stabilization of multiclass queuing networks. Master's thesis, IST, June 2001.

[5] Christhos G. Cassandras. *Discrete Event Systems: Modeling and Performance Analysis*. Asken Associates Incorporated Publishers, 1993.

[6] J. Michael Harrison. Brownian models of open processing networks: Canonical representation of workload. *The Annals of Applied Probability*, 10(1):75–103, 2000.

[7] J. G. Dai and Wuqin Lin. Maximum pressure policies in stochastic processing networks. *Operations Research*, 53(2):197–218, March-April 2005.

[8] K. Arnold, J.Gosling, and D. Holmes. *Java Programming Language*. Addison Wesley, fourth edition, 2005.

[9] P. L'Ecuyer. Efficiency improvement and variance reduction. In *Winter Simulation Conference*, pages 122–132, December 1994.

[10] Z. Fernandes. Sequenciamento em filas de servidor único. Variante da regra μc. Master's thesis, IST, Outubro 2007.

[11] M. Rebello, Z. Fernandes, and C. Bispo. Simulating activity networks in JAVA. 6th Eurosim Congress on Modelling and Simulation, 2007.

[12] Steve H. Lu and P. R. Kumar. Distributed scheduling based on due dates and buffer priorities. *IEEE Transactions on Automatic Control*, 36:1406–1416, 1991.