Controlo de Admissão Diferenciado e Adaptativo em Sistemas de Tempo Real

André Filipe dos Santos Ferreira[1,2], e-mail: andre-s-ferreira@ptinovacao.pt
Solange Rito Lima[2], e-mail: solange@di.uminho.pt

[1] Portugal Telecom Inovação, SA, 3810-1/6 Aveiro, Portugal

[2] Departamento de Informática, Universidade do Minho, 4710-057 Braga, Portugal

***Resumo* — Nas plataformas de prestação de serviços é necessária a presença de mecanismos de controlo de admissão para garantias de robustez e qualidade de serviço. Este artigo apresenta um mecanismo de admissão que controla e modela o ritmo com que os pedidos de requisição de serviço são admitidos numa plataforma. Na solução proposta são introduzidos conceitos de classe de serviço para diferenciação do processo de admissão segundo os perfis definidos. O mecanismo de controlo utiliza algoritmos de escalonamento inovadores que permitem fornecer garantias de qualidade de serviço em cenários de carga elevada, de um modo eficaz, justo e equitativo.**

1. Introdução

Numa plataforma de prestação de serviços os clientes invocam pedidos de serviços esperando que seja devolvida uma resposta num determinado período de tempo, que se pretende que seja o mais breve possível.

Este processo de comunicação é usualmente efectuado através de interfaces responsáveis pela invocação do serviço necessário para satisfação desses pedidos.

A execução dos pedidos consome, inevitavelmente, recursos de memória e processamento. Deste modo, se a admissão de pedidos à plataforma não for controlada, podem ocorrer períodos em que um número demasiado elevado de pedidos são admitidos, levando a um cenário de congestão e retenção de demasiados recursos. Em cenários críticos pode mesmo ser atingido um ponto de ruptura, onde a satisfação de serviços é impossibilitada na sua totalidade. Assim, a presença de um mecanismo de controlo de admissão é fundamental para garantir a robustez e qualidade de serviço prestada, evitando cenários de congestão resultantes de uma admissão indiscriminada e descontrolada de pedidos. Limitando o ritmo de aceitação dos pedidos controla-se, indirectamente, o nível de carga induzida na plataforma.

Um aspecto importante num mecanismo de controlo de admissão consiste na capacidade de diferenciar a ordem de atendimento dos pedidos. Um método eficaz de o fazer consiste na agregação de múltiplos serviços em classes de serviço para atendimento diferenciado e com prioridades distintas de admissão.

Neste âmbito, este artigo apresenta uma solução que permite o controlo e modulação do ritmo de admissão dos pedidos submetidos a uma plataforma de prestação de serviços, de uma forma flexível, diferenciada e configurável, permitindo a atribuição de prioridades distintas entre os vários serviços. São assim proporcionadas garantias de qualidade de serviço em cenários de elevada carga, onde o ritmo máximo de admissão é por vezes superado. A solução, desenvolvida em *Java EE*, foi implementada a nível aplicacional, numa interface de uma plataforma de prestação de serviços de um operador nacional.

A estrutura deste artigo encontra-se organizada do seguinte modo: na secção 2 é apresentado um estudo dos principais métodos de controlo de ritmo e dos mais relevantes algoritmos de escalonamento desenvolvidos até à actualidade. É realizada a comparação das várias vertentes que a solução pode adoptar, fundamentando as opções realizadas; na secção 3 é apresentada a solução desenvolvida, sendo descrita a arquitectura do mecanismo, o seu ambiente de integração e especificados alguns detalhes de desenvolvimento; na secção 4 são descritos os testes a que a soluçao proposta foi submetida e avaliados os seus resultados; na secção 5 são apresentadas as conclusões deste trabalho.

2. Estado da Arte

Um mecanismo de controlo de admissão diferenciado recorre a metodologias de controlo de ritmo e escalonamento, que deverão estar ajustadas a cada tipo de cenário de aplicação. Nesta secção, apresenta-se um estudo comparativo dos principais métodos de controlo de ritmo e disciplinas de escalonamento, que sustentam as decisões tomadas na solução proposta neste artigo.

Controlo de Ritmo

Um mecanismo de controlo do ritmo deve permitir a limitação e modelação do ritmo de expedição de pedidos com a granularidade desejada.

Um método simples de limitação de ritmo é designado *Leaky Bucket*. Este método recorre a uma fila de espera, designada *bucket*, que armazena os pedidos que lhe vão sendo inseridos, expedindo-os por ordem de chegada a uma taxa rígida especificada. Os pedidos que chegam quando o *bucket* está lotado são normalmente descartados.

Embora este algoritmo limite eficazmente a taxa com que os pedidos são enviados, é ineficiente nos casos em que a taxa limite raramente é atingida pelo ritmo de chegada. Para além desse facto, se não for recebida nenhuma informação durante um determinado período de tempo, a largura de banda residual não pode ser aproveitada para futuras transmissões.

Um método mais flexível consiste no *Token Bucket*, que possui uma filosofia semelhante ao *Leaky Bucket*, mas que permite uma regulação do ritmo mais flexível, permitindo a admissão de rajadas de pedidos consecutivamente. Este método utiliza unidades de crédito, designadas *tokens*. Normalmente para cada admissão é consumido um *token*, sendo a taxa de admissão determinada pela taxa com que os *tokens* são adicionados ao *bucket*. A quantidade de *tokens* permitidos depende do tamanho do *bucket*. Sempre que o *bucket* ficar vazio, o processo de admissão é suspendido até que novos

tokens sejam adicionados ao *bucket*. Estas características tornam o método *Token Bucket* a estratégia mais adequada para controlo do ritmo de admissão.

Disciplinas de Escalonamento

Um mecanismo de controlo de admissão diferenciado usualmente recorre a mecanismos de fila de espera e a escalonadores que devem efectuar o serviço das mesmas resolvendo o problema de contenção com equidade [4].

Para se efectuar diferenciação é necessário a distribuição da largura de banda de um modo justo e equitativo, proporcionalmente às prioridades definidas nas filas de espera, de modo a garantir-se que os recursos não são negados indefinidamente. A tentativa de optimização dos recursos atribuídos a uma fila activa pode comprometer as garantias de atendimento de filas activas de prioridades inferiores [5], o que é inaceitável na maioria dos cenários.

As disciplinas de escalonamento que pretendem atribuir garantias de atendimento e evitar a negação de recursos seja qual for o cenário de carga das filas, guiam-se por um modelo de escalonamento ideal, designado *Generalized Processor Sharing* (GPS) [9]. Neste modelo, o escalonamento é efectuado sobre fluxos contínuos, sendo atendidas quantidades infinitesimais de informação em simultâneo, de um modo justo para qualquer instante de tempo considerado. A diferenciação é suportada graças à capacidade de alocar diferentes níveis de largura de banda entre os fluxos activos.

A noção de infinitesimal, que considera fluxos contíguos de informação, infinitamente divisíveis, faz com que este modelo seja impossível de implementar, e sirva apenas de referência de ideal para as disciplinas de escalonamento implementáveis, que devido ao seu serviço iterativo, nunca atingem um nível de equidade perfeito.

Outras disciplinas, baseadas em *Round-Robin* (RR), estão normalmente associadas à baixa complexidade temporal devido à sua natureza estática, sendo úteis face a poucos recursos computacionais [7]. Estas disciplinas efectuam um escalonamento estático e cíclico.

Existem disciplinas derivadas do RR clássico, como o *Weighted Round Robin* (WRR), que efectua o serviço de vários pacotes consecutivamente, de modo a cumprir a relação proporcional ao peso definido para cada fila de espera. Para pacotes de tamanho variável é considerado o seu tamanho médio, sendo esta uma operação que induz complexidade no escalonamento [9]. A *disciplina Deficit Round Robin* (DRR) [10] elimina este problema através da utilização de um contador, designado *deficit counter*, que permite o serviço de pacotes de tamanho variável sem considerar o seu tamanho médio. Esta disciplina atinge melhor aproximação ao GPS que as disciplinas RR e WRR

Disciplinas baseadas em Fair Queue (FQ) [11], conseguem melhores aproximações ao ideal GPS. A disciplina FQ efectua o serviço de um modo proporcional sem atendimento cíclico, sem deixar de evitar a negação de recursos às filas de espera activas. O tamanho do pacote é considerado para que sejam servidas quantidades de informação proporcionais ao peso das filas. Para isso, são utilizadas funções de tempo virtual de entrada e saída, sendo os pacotes servidos por ordem crescente do tempo virtual de saída. As funções de tempo virtual tomam em consideração o tempo real de entrada somado ao tempo que o pacote teoricamente demora a ser atendido no ideal GPS. Exemplos de disciplinas FQ que permitem a diferenciação de serviço são a *Packet-by-Packet GPS* (PGPS) [9] e a *Weighted Fair Queuing* (WFQ) [12], sendo idênticas mas apresentadas independentemente.

Ambas são consideradas muito próximas do ideal GPS.

As funções de tempo virtual utilizadas nestas disciplinas contemplam o custo associado a cada fila, assim como o tamanho do pacote e um indicador do número de rondas efectuadas. Em [13] é demonstrado que a disciplina WFQ não está tão próxima do GPS como suposto, e é apresentado o algoritmo *Worst Case Fair Weight Fair Queuing* (WF2Q) que, mantendo os princípios WFQ, garante equidade mesmo no pior cenário. Contudo, a computação de funções de tempo virtual para cada pacote induz complexidade no processo de escalonamento.

Numa tentativa de reduzir a complexidade das disciplinas WFQ e W2FQ, mantendo as suas propriedades, é apresentado em [14] o algoritmo SCFQ. Recorre ao uso de função de tempo virtual, mas considera uma etiqueta temporal relacionada com o pacote servido anteriormente. Assim, o SCFQ tem a sua própria referência temporal, que depende exclusivamente do progresso das filas de espera servidas, sendo mais simples que o WFQ. Contudo, apesar de menor complexidade, o SCFQ é menos eficaz que o WFQ, podendo ser injusto em curtas prestações de serviço [13].

Em 2005, foi apresentado em [15] a disciplina *Most Credit First* (MCF), que utiliza um sistema de créditos que permite balancear o nível de serviço que cada fila deve receber segundo um modelo de equidade ideal com o nível de serviço que é efectivamente prestado pelo algoritmo. Em cada iteração, a fila com mais créditos disponíveis é atendida. O peso de cada fila também é considerado. A equidade proporcional ao peso das filas é atingida através da penalização das filas que possuem créditos negativos, não as atendendo até que os seus créditos aumentem. É demonstrado em [15] que o MCF tem melhores performances que o WFQ e o DRR.

Em [16] é apresentada a disciplina *Credit-Based Fair Queuing* (CBFQ), que recorre à utilização de indicadores para identificar a quantidade de créditos acumulados, que reflectem o rácio de atendimento de cada fila. Este algoritmo é adaptativo, considerando o tamanho dos pacotes e o nível de carga existente em cada fila de espera activa, iterativamente. Baseado nestas métricas o serviço é balanceado de acordo com as condições de carga ao longo do tempo.

As disciplinas MCF e CBFQ fornecem as vantagens de algoritmos que recorrem a funções de tempo virtual, evitando a sua elevada complexidade, sendo mais facilmente implementáveis, sem deixar de efectuar uma atribuição justa de largura de banda. Estas disciplinas revelam-se assim as mais apropriadas a implementar num mecanismo de admissão diferenciado.

3. Solução Desenvolvida

Nesta secção é apresentada uma solução de controlo de admissão diferenciado e adaptativo desenvolvida em *Java EE*, implementada a nível aplicaçional numa interface de acesso a uma plataforma de prestação de serviços.

A. Requisitos

A solução apresentada considera os seguintes requisitos:

- limitação da taxa de admissão com que os pedidos são encaminhados para a plataforma;
- possibilidade de armazenar os pedidos em situações em que o ritmo limite é superado, para posterior atendimento se possível;
- diferenciação justa, garantindo equidade de serviço em função da prioridade definida para cada classe de serviço;
- opção de serviço adaptativo, segundo a carga associada a cada classe de serviço, proporcionando a atribuição flexível de largura de banda ao longo do tempo;
- capacidade de configurar a granularidade com que a limitação do ritmo é efectuada, assim como os perfis que caracterizam as classes de serviço. As alterações devem ser aplicadas sem que a interface precise de ser reiniciada.

B. Ambiente de Integração

O mecanismo de admissão integra-se numa interface de acesso a uma plataforma de prestação de serviços. Esta interface permite a satisfação de pedidos de aprovisionamento *HTTP URL Encoded*, efectuados por entidades externas à plataforma.

A interface está desenvolvida em Java2EE, sendo compilada recorrendo ao Ant. É executada em plataformas que suportem os servidores aplicacionais *WebLogic* e *JBoss*.

A interface de acesso encontra-se dividida em dois níveis operacionais: o nível de interfaces, que estabelecem a ligação das entidades externas com o segundo nível, o *engine*, que possui toda a lógica de funcionamento e regras de acesso aos serviços.

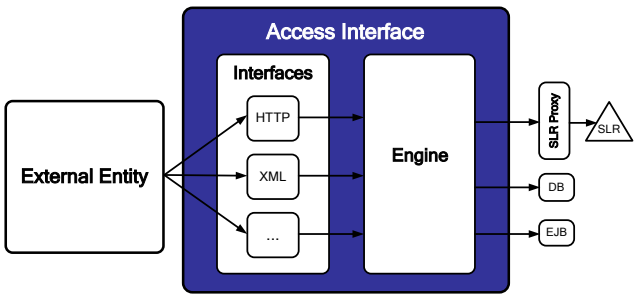

Figura 1. Arquitectura simplificada da Interface de Acesso

Os pedidos efectuados pelas entidades externas à interface de acesso sofrem um processo de interpretação e verificação de admissão antes de serem encaminhados para o *engine*. Inicialmente, os pedidos são encaminhados para a interface correspondente, sendo-lhes atribuída uma identificação. De seguida, o *engine* verifica se a entidade externa tem permissão para aceder ao serviço invocado. Também é verificado se a invocação do pedido é efectuada no período horário permitido, assim como se o número máximo de ligações permitidas à entidade externa associada não é excedida.

Se for atribuída permissão ao pedido, é então executado o serviço nele invocado, podendo este consistir na invocação de primitivas, procedimentos ou funções de base de dados, assim como em métodos existentes em *EJBs*. A resposta é finalmente entregue pelas interfaces à entidade externa, no formato XML. As entidades externas emitem os pedidos a um ritmo indeterminado. Assim, na transição do pedido das *interfaces* para o *engine*, antes da execução de cada pedido, estes são submetidos a um processo de controlo de admissão que efectua a autenticação e validação. Se for concedida permissão, o serviço invocado em cada pedido é executado, caso contrário, este não pode prosseguir e é devolvida à entidade externa uma reposta com uma mensagem esclarecedora do motivo.

C. Processo de Admissão

Focando mais detalhadamente o processo de admissão aplicado a cada pedido que ingressa na interfaçe, apresenta-se na Fig. 2 o procedimento efectuado.

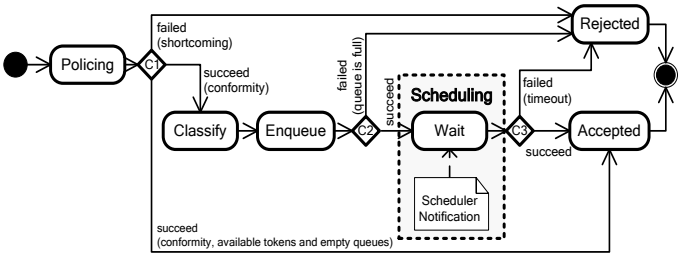

Figura 2. Algoritmo de controlo de admissão

Quando o pedido ingressa na interface é verificado se este possui permissão para aceder ao serviço que invoca e se este pode ser invocado no período actual, de acordo com o contrato especificado. De seguida é verificado se a admissão do pedido provoca a infracção do ritmo de admissão máximo (condição C1). Se tal não se verificar, o pedido é admitido e o seu processo de admissão termina. Caso contrário, se o ritmo se encontrar superado, é necessário armazenar uma referência do pedido numa fila de espera, sendo consultada a classificação do pedido segundo a classe de serviço associada, e inserida na fila de espera da classe de serviço respectiva. Se a fila de espera estiver lotada (condição C2), a referência não é inserida, o pedido não é admitido e o processo de admissão termina. Se contudo, a referência ao pedido foi inserida na fila de espera com sucesso, o pedido irá aguardar por uma notificação do escalonador, que indica que a referência foi servida. A notificação pode indicar se o pedido foi ou não admitido (condição C3). No caso do pedido ter sido servido normalmente o processo de admissão é concluído com sucesso, contudo, no caso em que o tempo limite de espera pelo serviço é atingido, o pedido é rejeitado, e o processo de admissão termina.

O processo de admissão é assim simplista no caso do ritmo médio de chegada dos pedidos ser inferior ao limite, sendo apenas efectuado todo o processo descrito nas situações em que o ritmo é superado. Deste modo os pedidos que seriam imediatamente descartados são armazenados para serem atendidos quando possível.

D. Arquitectura do Mecanismo

O processo de admissão descrito na secção B é efectuado sobre a arquitectura ilustrada na Figura 3.

Os pedidos ingressam na plataforma a ritmos imprevisíveis, sendo o mecanismo responsável pela sua admissão segundo uma taxa normalizada.

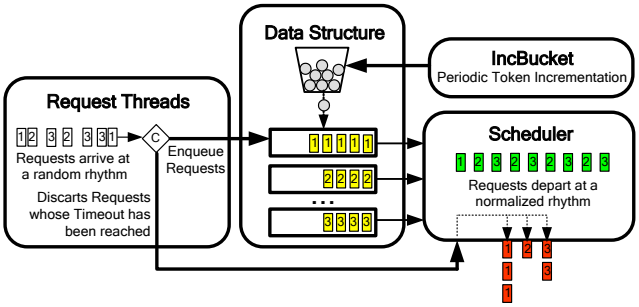

Figura 3. Arquitectura do mecanismo

A arquitectura é constituída por quatro módulos:

Estruturas de Dados: As estruturas de dados que suportam o controlo de admissão consistem num contador, designado *bucket*, e numa fila de espera para cada classe de serviço configurada.

Segundo a terminologia *Token Bucket*, o *bucket* define a granularidade e elasticidade com que os pedidos em fila de espera são servidos. Para isso, são configurados os parâmetros de limite das rajadas de pedidos (pedidos admitidos consecutivamente) e o número de unidades a incrementar periodicamente. Quando o *bucket* fica vazio é sinal que o ritmo de admissão permitido foi atingido, o que força a que o escalonador aguarde pela incrementação de mais unidades de *tokens* no *bucket*.

A utilização de filas de espera permite o armazenamento de referências de pedidos. Todas as referências existentes numa fila de espera são relativas a pedidos de uma mesma classe de serviço, possuindo iguais prioridades de serviço, não sendo necessária a ordenação das filas de espera. Assim, cada fila é servida segundo a disciplina *first-in first-out*.

Threads de Pedidos: O processo de admissão de cada pedido é efectuado numa *thread*, que após efectuar a fase de policiamento, classifica e insere o pedido na fila de espera respectiva. De acordo com o processo de admissão, quando o ritmo não é superado, i.e., quando o *bucket* não possui mais *tokens* disponíveis, o pedido é admitido imediatamente, evitando o recurso às filas de espera e respectivos escalonadores. Contudo, basta existir um elemento em qualquer uma das filas de espera para que a admissão directa dos pedidos não seja permitida, sendo atribuída prioridade ao atendimento das filas de espera. Em cenários em que o ritmo é superado e as referências aos pedidos são inseridas nas filas de espera, a thread aguarda a notificação do escalonador com a respectiva resposta de admissão.

Incrementador: O incrementador é uma *thread* responsável pela adição periódica de unidades no *bucket*. A taxa de incremento pode ser alterada variando o número de unidades adicionadas em cada iteração. Quantas mais unidades forem incrementadas por iteração, maior é o intervalo de tempo por iteração, ocorrendo menos iterações por segundo, contudo a perfeita granularidade do ritmo de aceitação é sacrificada, passando os pedidos a ser admitidos em pequenas rajadas. Assim, o incrementador é responsável pela modulação do ritmo de admissão.

Escalonador: O escalonador é responsável por garantir que o ritmo de admissão não é superado, servindo iterativamente as referências dos pedidos segundo o algoritmo seleccionado. O escalonador serve as filas de espera sempre que existirem unidades a consumir no *bucket*. Para cada referência que é atendida da fila de espera, o *bucket* é decrementado uma unidade. O escalonamento pode ser efectuado segundo as duas disciplinas de escalonamento implementadas (MCF ou CBFQ). O MCF [16] deve ser utilizado em cenários onde se pretende a atribuição justa de largura de banda com um comportamento linear, devendo ser o CBFQ [17] utilizado para um escalonamento adaptativo, consoante as condições de carga das filas.

Coordenação: As operações efectuadas pelos vários módulos são executadas de modo paralelo, mas dependem umas das outras. Este facto levanta a necessidade de uma coordenação cuidada entre as operações. Existem três níveis de coordenação:
(i) no caso das filas de espera se encontrarem vazias o escalonador pára, até que um novo pedido chegue, sendo emitida uma notificação da *thread* do pedido para o escalonador arrancar de novo o seu atendimento;
(ii) no caso de não existirem unidades disponíveis no *bucket* o escalonador pára, até que sejam incrementadas unidades no *bucket*, sendo emitida uma notificação do incrementador para o escalonador arrancar de novo o seu atendimento;
(iii) sempre que o escalonador serve a referência de um pedido, envia uma notificação para a *thread* do pedido respectivo, que se encontra à espera de uma notificação que traduza a resposta de admissão do pedido;
(iv) sempre que o *bucket* está cheio e não chegam novos pedidos, o incrementador suspende, até que uma notificação seja recebida por parte do escalonador, indicando a necessidade de activar o incrementador de novo.

A arquitectura apresentada pode ser aplicada não só em plataformas de prestação de serviços, como em outros tipos de módulos que necessitem controlo de admissão.

Tal processo é facilitado pela arquitectura modular da solução, e pela referência de cada pedido consistir num simples identificador. A comunicação com a interfaçe só ocorre ao nível das *threads* relativas a cada pedido. Deste modo, o mecanismo é mais facilmente implementado noutras soluções desenvolvidas em Java, sendo invocado no instante em que os objectos a coordenar e controlar são recebidos no módulo ou plataforma.

C. Detalhes de Desenvolvimento

O mecanismo de controlo de admissão foi primeiramente implementado de forma modular como protótipo, tendo sido integrado na interface de acesso através da invocação de um método que executa o controlo de admissão de cada pedido. São brevemente descritos de seguida algumas características da solução desenvolvida.

Logs

As operações que decorrem ao longo do processo de admissão são registadas em *logs*. A quantidade de *logs* gerados depende do modo como o serviço de *logs* se encontra configurado, segundo os níveis hierárquicos: *debug*, *info*, *warn*, *error*, e *fatal*. Apenas o resultado do controlo de admissão de cada pedido gera *logs* do tipo *info*, sendo os restantes do tipo *debug* ou *error*.

Códigos e mensagens de erro

Em caso de erro, são devolvidos códigos às interfaces, que por sua vez mapeiam o código com a mensagem de

erro que deve ser devolvida à entidade que invocou o pedido. Encontram-se definidos dois códigos de erro para o âmbito do controlo de admissão: o código devolvido no caso de um pedido exceder o tempo de espera definido para uma resposta de admissão, e o código devolvido no caso de as filas de espera se encontrarem cheias, não sendo possível a execução do pedido em ambos os casos.

Em cada interface, existe um ficheiro responsável por mapear o código de erro com a resposta que deve ser devolvida pela interface respectiva.

Base de dados

A interface de acesso recorre a uma base de dados que armazena os parâmetros de configuração necessários segundo o diagrama representado na Fig. 3.

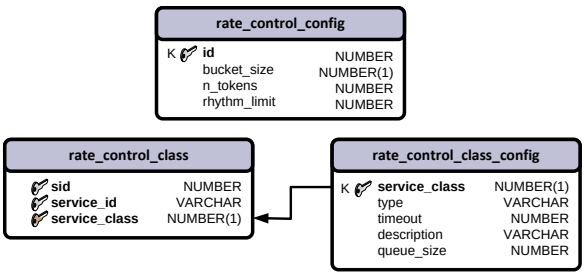

Figura 4. Tabelas de configuração do controlo de admissão

A tabela *AC_class* indica qual a classe de serviço associada a cada tuplo (identificação de serviço, identificação da sessão). A tabela *AC_class_config* possui as configurações associadas a cada classe de serviço. A tabela *AC_control_config* contém as configurações que permitem afinar o comportamento do mecanismo de controlo de ritmo.

Servidor Aplicacional

Os ficheiros binários que resultam da compilação da interface de acesso são depositados num servidor aplicacional responsável pela sua execução.

O servidor aplicaçional carrega a informação da base de dados através de *entityBeans*, armazenando-os em memória por determinado período de tempo, o que permite que a classificação seja efectuada mais rapidamente, sem acesso constante à BD. Deste modo, qualquer alteração ao nível da base de dados, pode ser aplicada à interface sem que esta necessite de ser reiniciada, bastando para isso limpar os dados armazenados na *cache* do servidor aplicacional.

3. Testes e Resultados

De modo a demonstrar o funcionamento da solução foram efectuados dois testes de carga, através da submissão concorrente de pedidos fictícios à interface.

Foram definidas três classes de serviço, associadas a três filas de espera, com prioridades de 50%, 30% e 20% respectivamente. Foram parametrizados três serviços, exclusivamente para testes, para três classes de serviço respectivamente. Estes serviços invocam um procedimento PL/SQL responsável por registrar a classe de serviço e o instante em que cada pedido é executado.

Esses registos são armazenados numa tabela da base de dados que vai permitir a geração de histogramas que apresentam o número de pedidos atendidos por segundo para cada classe de serviço. Em ambos os testes o mecanismo foi configurado para limitar o ritmo de admissão a 100 pedidos por segundo, e o timeout foi definido de modo a que nenhum pedido fosse rejeitado.

Teste 1: Diferenciação

Neste teste são emitidos concorrentemente 1200 pedidos de cada classe de serviço, com intervalos de 4 segundos entre a emissão de cada rajada de cada classe de serviço. O *bucket* foi limitado a uma unidade, para não ser permitida a admissão de rajadas de pedidos, estando a solução configurada para um controlo de ritmo rígido e de granularidade perfeita. O facto de o ritmo se encontrar limitado a 100 pedidos por segundo leva a que, quando pedidos de classe 2 e 3 são submetidos, ainda existam pedidos de classe 1 a aguardar atendimento. Deste modo pode ser verificada a equidade da partilha de largura de banda ao longo do tempo. O teste é efectuado duas vezes, com os algoritmos MCF e CBFQ.

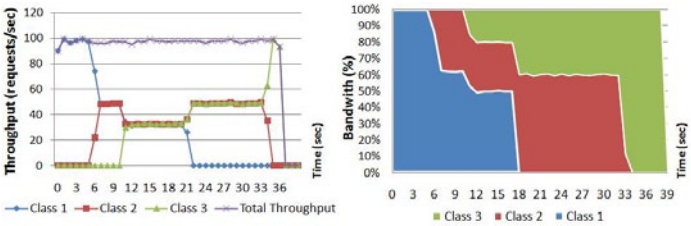

Figura 5. Resultados MCF

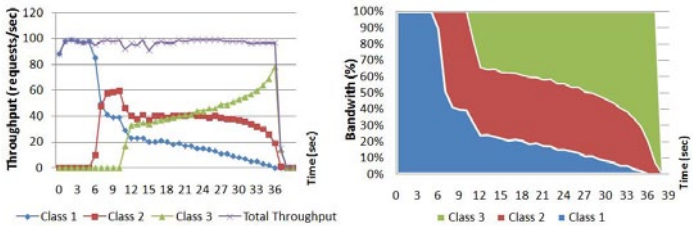

Figura 6. Resultados CBFQ

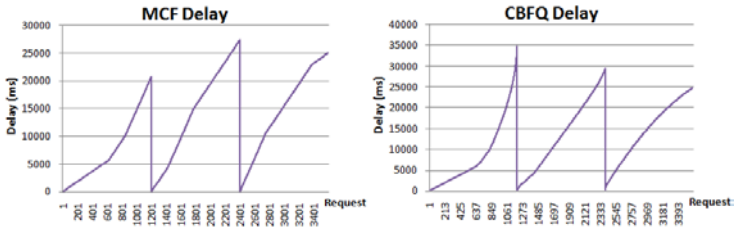

Figura 7. Atrasos de Serviço

Os resultados do teste 1 apresentam-se nas figuras 5, 6 e 7, que ilustram as taxas de pedidos admitidos por segundo, a largura de banda consumida ao longo do tempo, e o atraso sofrido pelos pedidos em fila de espera.

Nos resultados do algoritmo MCF (Figura 5) observa-se que nos instantes iniciais, quando apenas existem pedidos de classe1, esta classe consome toda a largura de banda disponibilizada, atingindo a taxa de admissão máxima.

No instante em que os pedidos de classe2 são emitidos, o escalonador passa a distribuir a largura de banda pelas duas classes, proporcionalmente à sua prioridade. Analogamente, quando os pedidos de classe 3 são emitidos a largura de banda passa a ser distribuída pelas três classes de modo proporcional às suas prioridades.

Quando os pedidos de classe1 são totalmente satisfeitos, a largura de banda libertada por esta classe é distribuída pelas classes 2 e 3. Finalmente quando os pedidos de class2 são satisfeitos, a classe3 passa a disponibilizar da totalidade de largura de banda, não existindo concorrência.

Este teste demonstrou claramente o perfil de equidade prestado pelo escalonador MCF.

Nos resultados do algoritmo CBFQ (Figura 6) observa-se que nos instantes iniciais os pedidos de classe1 consomem toda a largura de banda disponibilizada, atingindo a taxa de admissão máxima. Contudo, em vez de a largura de banda ser distribuída tendo em conta apenas a prioridade de cada classe, é também tido em conta o nível de carga de cada fila de espera ao longo do tempo. A curva crescente observada no serviço da classe3 é devida ao facto de, embora a prioridade da classe 3 ser inferior, o seu nível de carga é superior em relação às restantes filas. Assim, para balancear o nível de carga das várias filas, a fila da classe3 é servida com maior prioridade, dadas as circunstâncias. Este facto torna o CBFQ apropriado para lidar com cenários de congestão onde as classes de serviço possuem uma afluência desproporcional que pode conduzir à negação de serviço das classes de menor prioridade.

Os atrasos apresentados na Figura 7 ilustram três variações relativas às classes 1, 2 e 3 respectivamente.

Relativamente aos resultados do algoritmo MCF, o motivo pelo qual os pedidos da classe2 possuem atrasos superiores à classe3 reside no facto da classe 2 concorrer constantemente pelo serviço, enquanto que a classe 3 dispõe de um período de tempo em que usa a totalidade de largura de banda. Os pedidos de classe1 têm menor atraso pois a classe1 possui maior prioridade.

Comparativamente, os resultados do algoritmo CBFQ demonstram que os pedidos de classe3 possuem os atrasos inferiores às restantes classes. Tal reside no facto de a classe3 adquirir maior prioridade, devido à existência de maior carga na sua fila de espera. Os atrasos da classe1 são os superiores, pois enquanto o balanceamento da carga é efectuado, o serviço desta classe sofre atrasos resultantes da atribuição de prioridade às filas que possuam um nível de carga superior. Assim, o CBFQ induziu atraso na classe de maior prioridade para poder reduzir a carga das classes de menor prioridade.

Teste 2: Comportamento a rajadas de pedidos

Neste teste são emitidos concorrentemente 900 pedidos de uma só classe de serviço, a um ritmo de 110 pedidos por segundo. O teste foi realizado três vezes com o *bucket* limitado a uma, dez e vinte unidades respectivamente. Como o ritmo de admissão está limitado a 100 pedidos por segundo, pretende-se demonstrar a flexibilidade que a permissão de rajadas de pedidos fornece.

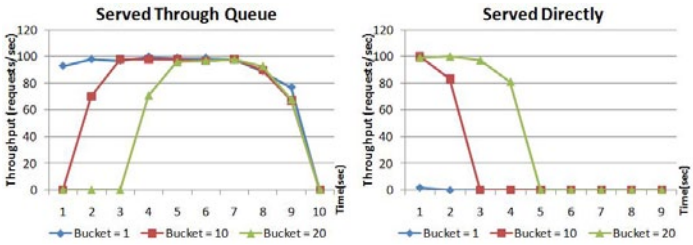

Figura 8. Comportamento a rajadas de pedidos

Observa-se que quanto maior é a capacidade do *bucket*, maior é a capacidade de encaminhar pedidos directamente e menor é a tendência de os servir através das filas de espera. Apesar da granularidade do ritmo ter sido sacrificada, o aumento do número de pedidos encaminhados directamente implica a redução do número de pedidos encaminhados através da fila de espera, reduzindo as iterações de escalonamento das filas de espera.

5. Conclusão

Este artigo apresentou uma solução de controlo de admissão que implementa os algoritmos de escalonamento recentes e inovadores Most Credit First (MCF) e Credit Based Fair Queuing (CBFQ).

A aplicação desta solução numa interface de acesso de uma plataforma de prestação de serviços permitiu a diferenciação de atendimento entre os serviços invocados pelos vários pedidos, através da introdução do conceito de classe de serviço. Deste modo, a regulação de ritmo de admissão dos pedidos que invocam os vários serviços considera os perfis definidos para as classes de serviço respectivas, com a granularidade desejada.

A solução foi sujeita a testes de carga, cujos resultados revelam a eficácia e robustez da mesma, cumprindo os requisitos definidos e proporcionando qualidade de serviço essencial para enriquecer a fiabilidade do processo de admissão. Os resultados obtidos nos testes efectuados permitiram confirmar que o comportamento da disciplina de escalonamento MCF é baseado na prioridade associada a cada fila de espera, sem considerar o nível de carga das mesmas. Confirma-se também que a disciplina CBFQ efectua o escalonamento tendo em consideração o nível de carga existente em cada fila de espera, mantendo um atendimento justo entre as mesmas.

Como trabalho futuro pretende-se analisar o comportamento de ambas as disciplinas de escalonamento utilizadas face a cenários de teste distintos. A solução desenvolvida poderá ainda ser alvo de optimização a nível de implementação, de modo a aumentar a sua eficiência.

Referências

[1] Briscoe, B., "*Flow rate fairness: dismantling a religion.*", SIGCOMM . Rev., 2007. 37(2): p. 63-74.

[2] Rexford, J.L., A.G. Greenberg, and F.G. Bonomi, "*Hardware-Efficient Fair Queueing Architectures for High-Speed Networks*", in *In Proceedings of INFOCOM*. 1996. p. 638-646.

[3] Wong, W.K., H.Y. Tang, and V.C.M. Leung, "*Token bank fair queuing: a new scheduling algorithm for wireless multimedia services: Research Articles*". Int. J. Commun. Syst., 2004. 17(6): p. 591-614.

[4] Boudec, J.-Y., "*Rate adaptation, Congestion Control and Fairness: A Tutorial*". 2000.

[5] Eryilmaz, A. and R. Srikant, "*Fair resource allocation in wireless networks using queue-length-based scheduling and congestion control*". IEEE/ACM Trans. Netw., 2007. 15(6): p. 1333-1344.

[6] Jalali, A., R. Padovani, and R. Pankaj, "*Data throughput of CDMA-HDR a high efficiency-high data rate personal communication wireless system*". Vehicular Technology Conference Proceedings, 2000. 3: p. 1854--1858.

[7] Hahne, E.L., "*Round-Robin Scheduling for Max-Min Fairness in Data Networks*". IEEE Journal, 1991. 9: p. 1024--1039.

[8] Trajkovic, N.A.a.B.C.a.L., "*Modeling Packet Scheduling Algorithms in IP Routers*". 2001: School of Engineering Science, Simon Fraser University.

[9] Parekh, A.K. and R.G. Gallager, "*A generalized processor sharing approach to flow control in integrated services network: the single node case*", in IEEE/ACM Trans. Netw. *(Vol. 1)*. 1993, IEEE Press

[10] Shreedhar, M. and G. Varghese, "*Efficient fair queueing using deficit round robin*", in SIGCOMM'95. 1995, ACM: Cambridge, Massachusetts, U.S.

[11] Nagle, J.B., "*On packet switches with infinite storage*", in *Innovations in Internetworking*". 1988, Artech House, Inc. p. 136-139.

[12] Demers, A., S. Keshav, and S. Shenker, "*Analysis and simulation of a fair queueing algorithm*". SIGCOMM Comput. Commun. Rev., 1989.

[13] Bennett, J.C.R., "*Wf2q : Worst-case Fair Weighted Fair Queueing*". 1996.

[14] Golestani, S.J., "*A self-clocked fair queueing scheme for broadband applications*". NFOCOM apos;94. , 1994. 2.

[15] Pan D., Y.Y., "*Credit based fair scheduling for packet switched networks*". INFOCOM 2005, 2005. 2.

[16] Bensaou, B., D.H.K. Tsang, and K.T. Chan, "*Credit-based fair queueing (CBFQ): a simple service-scheduling algorithm for packet-switched networks*". IEEE/ACM Trans. Netw., 2001. 9(5): p. 591-604.