Provisão de QoS Adaptável em Sistemas Operacionais: O Subsistema de Rede[1]

Marcelo F. Moreno, Antônio T. A. Gomes, Sérgio Colcher, Luiz F. G. Soares

Departamento de Informática – Pontifícia Universidade Católica do Rio de Janeiro
R. Marquês de São Vicente, 225, RDC – 22453-900 – Rio de Janeiro RJ – Brasil
{moreno, atagomes, colcher, lfgs}@inf.puc-rio.br

Abstract. *The progressive demand for distributed multimedia applications makes evident the need for end-to-end quality of service (QoS) provisioning. Particularly, operating systems must guarantee that the resources under their control are managed to fulfill the requirements of each application, whether they are located at end systems, switches or routers. This work proposes the definition of an architecture for QoS provisioning on operating systems, focusing on the packet queuing subsystem. The instantiation of the architecture is based on a few modifications to the standard Linux kernel, adding some other desirable features such as runtime service adaptation.*

Resumo. *A demanda progressiva por aplicações multimídia distribuídas torna evidente a necessidade de provisão de qualidade de serviço (QoS) fim-a-fim. Em particular, sistemas operacionais devem garantir que os recursos sob seu controle sejam gerenciados de modo a preencher as necessidades de cada aplicação, seja nas estações finais, comutadores ou roteadores. Este trabalho propõe a definição de uma arquitetura para provisão de QoS em sistemas operacionais, focalizando o subsistema de enfileiramento de pacotes. A instanciação da arquitetura é baseada em pequenas alterações do kernel padrão do Linux, adicionando outras características desejáveis como a possibilidade de adaptação de serviços em tempo de operação.*

1. Introdução

A crescente demanda por aplicações multimídia distribuídas torna evidente a necessidade de mecanismos que objetivem maximizar o uso dos recursos presentes no seu ambiente de operação (que inclui as estações finais e o provedor de serviços de comunicação) e que permitam oferecer a QoS fim-a-fim desejada pelos usuários. Dessa forma, a provisão de QoS requer a implementação de uma série de funções nas estações finais e no provedor de serviços. Nas estações finais, recursos controlados pelo sistema operacional, como CPU, memória e buffers de comunicação, devem ser gerenciados de

[1] Este trabalho foi realizado com apoio do Fundo Setorial para o Desenvolvimento Tecnológico das Telecomunicações (FUNTTEL), através do contrato 0594/02.

forma a assegurar que a coexistência de várias aplicações não viole as necessidades individuais de QoS. No provedor de serviços, os sistemas operacionais de cada comutador ou roteador devem prover as mesmas funcionalidades, além do gerenciamento dos canais de comunicação nas suas diversas portas de entrada e saída.

O desafio de prover QoS às aplicações torna-se ainda maior à medida que surgem novos requisitos em função de novos tipos de aplicações, novas técnicas de codificação etc. De fato, a implantação rápida de serviços com novas características de QoS tem se mostrado essencial para operadores de telecomunicações. Conforme mencionado em [Kosmas97], adaptações a novas demandas de QoS devem ocorrer, idealmente, por meio de pequenas modificações na infra-estrutura de comunicação e processamento. A especificação de um novo serviço pode envolver a escolha de algoritmos de escalonamento, admissão e classificação. Outros parâmetros de configuração podem ser citados, como as tarefas que irão compor a pilha de protocolos de comunicação ou a descrição do estado inicial do sistema para a provisão de QoS (e.g. particionamento inicial dos recursos para cada classe de aplicação). Visando esses e outros objetivos, várias abstrações de adaptabilidade foram propostas na literatura, tais como reflexividade, sinalização aberta, programação orientada a aspectos e redes ativas [Campbell99]. Essas abstrações dependem diretamente de que comutadores, roteadores e estações finais sejam explicitamente programáveis durante a provisão dos serviços, o que demanda um sistema operacional suficientemente flexível para tal.

Sistemas operacionais de código aberto permitem algumas adaptações por meio da modificação do código do *kernel* (núcleo), ou pela simples reconfiguração dos elementos nele presentes; ou ainda, através de atualização fornecida pelo distribuidor do software. Geralmente, essas ações são seguidas da recompilação do código do *kernel* ou, no mínimo, da reiniciação do sistema. Esse tipo de adaptabilidade em tempo de projeto é satisfatória apenas quando a demanda por novos serviços é baixa. A adaptabilidade em tempo de operação, por sua vez, possibilita o atendimento a uma demanda maior por reconfigurações e aperfeiçoamentos do sistema operacional, já que o *kernel* passa a ser dotado de grande dinamismo. Sistemas operacionais baseados na arquitetura de *microkernel* [Coulson96] disponibilizam grande parte de suas funcionalidades para adaptação em tempo de operação. Porém, tais sistemas são pouco populares, o que é particularmente desvantajoso em termos de sua aplicação em estações finais.

Por outro lado, um estudo sobre os sistemas operacionais de propósito geral (GPOS)[2] evidencia algumas das barreiras impostas à adaptabilidade do sistema e à provisão de QoS, conseqüências do processamento por compartilhamento do tempo e da estrutura monolítica em que se baseiam. Os escalonadores de processos da maioria dos GPOS recorrem ao uso de prioridades para privilegiar a execução de algumas aplicações em detrimento de outras. Seus subsistemas de rede são guiados por interrupções, o que pode causar anomalias no escalonamento [Druschel96]. Normalmente, as filas de transmissão de pacotes são compartilhadas entre as aplicações, não havendo meios para a classificação ou priorização dos pacotes. Por último, são poucos os mecanismos para a reserva de recursos e introdução de partes adaptáveis no *kernel* em tempo de execução.

[2]Os GPOS (General Purpose Operating Systems) são os sistemas encontrados comercialmente com certa popularidade e que não se dedicam a tarefas e aplicações específicas.

Um dos aspectos principais a serem abordados neste trabalho é como oferecer suporte adequado à provisão de QoS e à adaptabilidade em GPOS. Para isso, é apresentada uma arquitetura genérica para provisão de QoS em sistemas operacionais, denominada QoSOS. O desenvolvimento dessa arquitetura seguiu-se à análise de algumas soluções apresentadas na literatura e à percepção de semelhanças funcionais entre elas. A arquitetura QoSOS permite reutilizar funções comuns e definir uma organização interna que seja equivalente nos diferentes sistemas, facilitando a definição de mecanismos de orquestração dos recursos do sistema geral como um todo.

A arquitetura QoSOS foi definida a partir da especialização e extensão dos *frameworks* para provisão de QoS em ambientes genéricos de processamento e comunicação, apresentados em [Gomes01]. Esses *frameworks* identificam conjuntos de funções recorrentes de provisão de QoS em vários subsistemas, como redes de comunicação, sistemas operacionais e plataformas distribuídas. Em [Gomes01] é demonstrado também como essas funções participam da orquestração de recursos para o fornecimento de serviços com QoS verdadeiramente fim-a-fim. A estruturação sob a forma de *frameworks* visou facilitar a identificação dos pontos de flexibilização (*hot-spots*) que devem ser preenchidos para descrever a funcionalidade de um ambiente específico.

Particularmente, o presente trabalho mostra como alguns desses *hot-spots* podem ser especializados para acomodar técnicas que visem a provisão de QoS especificamente em sistemas operacionais. Alguns *hot-spots* são, no entanto, deixados em aberto para que a arquitetura QoSOS atenda aos requisitos de adaptabilidade a novos serviços, possibilitando a configuração de funções como escalonamento e controle de admissão. Este trabalho também estende o trabalho apresentado em [Gomes01], incorporando à arquitetura QoSOS um modelo que permite a adaptação automatizada de determinados *hot-spots*, observando questões como manutenção de consistência.

Ilustrando um cenário de uso para a arquitetura QoSOS, o presente trabalho mostra também que o sistema *Linux*, apesar de caracterizado pelas desvantagens comuns aos GPOS, pode ser modificado para oferecer suporte a mecanismos adaptáveis de provisão de QoS. O enfoque deste artigo está sobre o subsistema de controle de tráfego do *Linux*, que oferece funcionalidades básicas para a reordenação de pacotes e policiamento dos fluxos nos buffers de comunicação em rede. Porém, é importante ressaltar que a instanciação da arquitetura QoSOS apenas sobre os buffers de comunicação não é uma solução completa para o problema da QoS em sistemas operacionais, ficando claro, na Seção 2, que outros recursos devem participar da orquestração, em especial a CPU.

Este artigo está organizado da seguinte forma. A Seção 2 descreve a arquitetura QoSOS. A Seção 3 apresenta a instanciação da arquitetura sobre o subsistema de rede do sistema operacional *Linux*. A Seção 4 contextualiza o presente trabalho junto a alguns estudos relacionados. Por fim, a Seção 5 é destinada às conclusões e trabalhos futuros.

2. Descrição da arquitetura

A arquitetura QoSOS foi definida a partir da especialização dos *frameworks* genéricos descritos em [Gomes01], acrescidos de algumas novas estruturas aqui introduzidas. A Figura 1 mostra como os tipos de *hot-spots* descritos em [Gomes01] podem ser completados para a construção de uma arquitetura de provisão de QoS em sistemas operacionais. Os *frameworks* genéricos definem as estruturas comuns para a provisão de

QoS nos vários subsistemas que participam do fornecimento do serviço fim-a-fim. A primeira etapa de especialização é feita para que sejam incluídas funcionalidades específicas de sistemas operacionais, como os mecanismos pertinentes aos subsistemas de escalonamento de processos e de comunicação em rede. Na etapa seguinte, são definidos os aspectos relacionados à provisão do serviço, como o conjunto de políticas de QoS que cada um dos subsistemas disponibilizará a seus usuários.

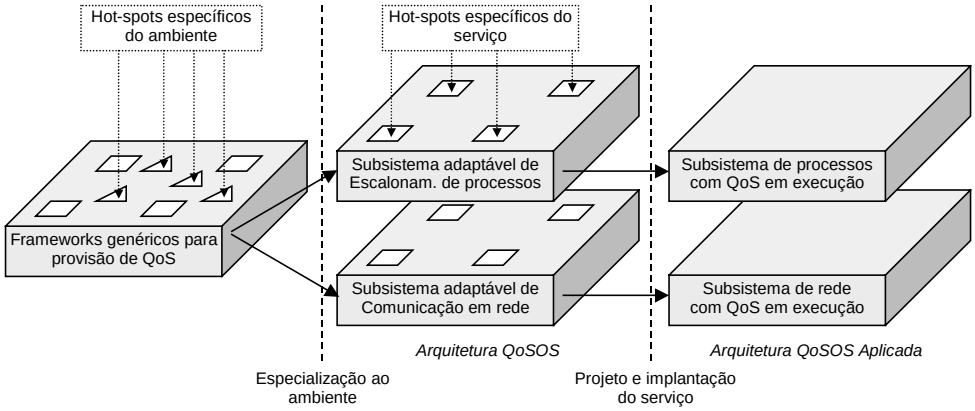

Figura 1 - Hot-spots da arquitetura modelada a partir dos frameworks genéricos

A descrição da arquitetura apresentada a seguir se encontra em formato textual e resumido. A descrição completa pode ser encontrada em [Moreno02].

2.1. Framework para parametrização de serviços

O *framework* para parametrização de serviços modela a estrutura responsável por definir um esquema de parâmetros de caracterização de serviços de forma genérica, independente dos possíveis serviços a serem oferecidos pelo sistema operacional. Tais parâmetros descrevem o comportamento tanto dos fluxos dos usuários quanto dos subsistemas que compõem o sistema operacional. Nota-se que ambas as caracterizações são fornecidas a partir da solicitação do serviço por parte do usuário, quando são informados os requisitos de processamento e comunicação desejados (parâmetros de especificação de QoS) e a dinâmica de geração dos seus dados (parâmetros de caracterização de carga). Um terceiro conjunto de parâmetros é utilizado para descrever as informações sobre o estado interno de cada subsistema, como, por exemplo, a quantidade de recursos disponíveis aos usuários (parâmetros de desempenho do provedor). Esse estado interno de cada subsistema é alterado a cada admissão de um novo fluxo, ou quando o subsistema sofre algum tipo de adaptação.

É importante observar que cada subsistema que compõe a infra-estrutura de provisão de serviços pode possuir uma forma distinta de descrição dos parâmetros citados. Por exemplo, a especificação de QoS para o subsistema de escalonamento de processos pode ser descrita por parâmetros como percentagem de uso da CPU, número de instruções a serem executadas em um intervalo ou o par quantum/período de execução. Já o subsistema de rede de um sistema operacional pode oferecer parâmetros como largura de banda, retardo máximo e taxa de perda de pacotes. Além disso, diferentes níveis de abstração também podem se utilizar de diferentes tipos de descrição e parâmetros, cabendo aos mecanismos atuantes durante a negociação de QoS do sistema operacional o mapeamento dos parâmetros de um nível de abstração superior para os

parâmetros que descrevem o comportamento dos recursos. A hierarquia de parâmetros oferecida pelo *framework* possibilita a criação de parâmetros abstratos, que devem ser especializados conforme as circunstâncias particulares, promovendo a generalidade necessária para a definição de parâmetros em diferentes níveis de abstração.

Categorias de serviço agrupam conjuntos de parâmetros que possuem características em comum, relacionadas ao tipo de ambiente, nível de visão de QoS, tipos de dados e de necessidades do usuário. A possível associação das políticas às categorias de serviço simplifica a ação dos mecanismos, já que a manipulação dos parâmetros ocorrerá no contexto da categoria desejada.

2.2. Frameworks para compartilhamento de recursos

Os *frameworks* para compartilhamento de recursos se baseiam no conceito de *recurso virtual* para modelar os mecanismos de alocação e escalonamento. Recursos virtuais são parcelas de utilização de um ou mais recursos reais distribuídas entre os fluxos submetidos pelos usuários. Para facilitar o emprego de vários algoritmos de escalonamento sobre um mesmo recurso e, assim, oferecer um conjunto amplo e flexível de serviços em um mesmo sistema, os recursos virtuais são dispostos em uma estrutura chamada de *árvore de recursos virtuais*. Cada recurso real possui uma árvore de recursos virtuais associada, embora uma mesma árvore de recursos possa representar a estrutura de escalonamento sobre mais de um recurso real. Um exemplo de árvore sobre vários recursos é o escalonamento de processos em sistemas multiprocessados.

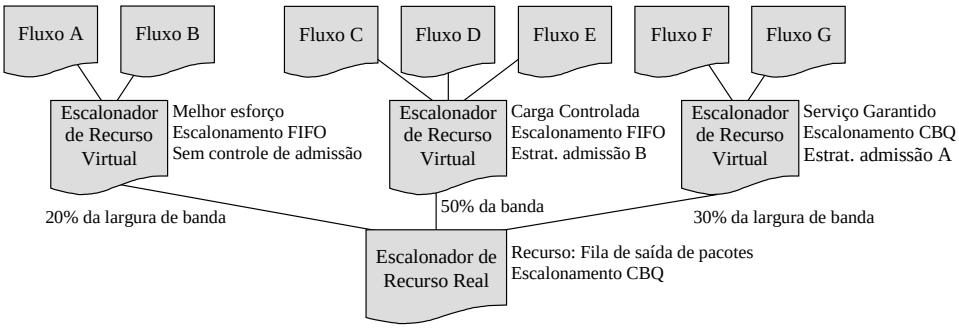

Figura 2 – Exemplo de árvore de recursos virtuais para um canal de comunicação, na provisão de serviços integrados

Em uma árvore de recursos virtuais, as folhas representam os recursos virtuais. A raiz da árvore corresponde ao escalonador de mais baixo nível da hierarquia, aquele que realmente distribui o tempo de uso do recurso real entre os nós filhos. Adicionalmente, este escalonador, denominado escalonador de *recurso raiz*, pode permitir que os recursos virtuais filhos utilizem diferentes recursos reais de um mesmo tipo. Os nós intermediários da árvore são *recursos virtuais especializados*, responsáveis por ceder a sua parcela de utilização do recurso real aos seus recursos virtuais filhos. Denominados *escalonadores de recursos virtuais*, esses nós podem ter acesso a mais de um recurso real por vez, para também permitir que seus nós filhos sejam atendidos simultaneamente. A Figura 2 ilustra um exemplo de árvore de recursos virtuais.

Cada escalonador de recurso virtual está associado a uma categoria de serviço e às políticas de provisão de QoS correspondentes, como as estratégias de escalonamento e de admissão, além de um componente de criação de recursos virtuais. Para efetivar a

criação de um recurso virtual, esse componente executa tarefas como a adição do recurso virtual à lista de responsabilidades do escalonador e a configuração dos módulos de classificação e de policiamento.

2.3. Frameworks para orquestração de recursos

Na área de atuação dos sistemas operacionais, vários são os recursos que devem ter seus mecanismos de escalonamento e de alocação gerenciados de forma integrada, em um processo de orquestração dos recursos de todo o ambiente. Na arquitetura QoSOS, a modelagem da orquestração de recursos é apresentada pela especialização de dois *frameworks distintos*: o *framework* para negociação de QoS e o *framework* para sintonização de QoS.

O *framework* para negociação de QoS modela os mecanismos de negociação e mapeamento que operam durante as fases de solicitação e estabelecimento de serviços, além dos mecanismos de admissão que atuam somente na fase de estabelecimento. Já o *framework* para sintonização de QoS modela os mecanismos de sintonização e monitoração que atuam na fase de manutenção do serviço.

Ao receber uma requisição de serviço, o controlador de admissão do sistema operacional deve verificar a viabilidade de aceitação do serviço naquele nível de abstração. Para isso, ele repassa a requisição ao agente de orquestração do sistema operacional, responsável por identificar todos os recursos reais que podem estar envolvidos no fornecimento do serviço e, então, distribuir entre eles as parcelas de responsabilidade sobre a provisão da QoS especificada. No caso de uma aplicação distribuída, o orquestrador do sistema operacional identificará que CPUs e buffers de comunicação são recursos que devem participar do oferecimento do serviço. A negociação de QoS em um sistema operacional é feita de forma centralizada, já que um único agente pode ter o conhecimento sobre todos os recursos do ambiente.

As aplicações multimídia distribuídas devem ter garantidas as suas necessidades sobre cada uma das *threads* que compõem seus processos, bem como sobre as *threads* que executam a pilha de protocolos. Além disso, os buffers de comunicação compartilhados devem ser capazes de encaminhar os pacotes de acordo com a parcela de QoS atribuída à estação pelo protocolo de negociação de rede e, por isso, a forma de implementação do subsistema de rede deve ser considerada na distribuição das responsabilidades. As estratégias de negociação definem como será a política de orquestração, baseando-se na implementação de subsistemas específicos.

A partir das parcelas de responsabilidade atribuídas a cada recurso, são acionados os mecanismos de mapeamento, consistindo na tradução da categoria de serviço (e dos parâmetros associados), especificada na solicitação do serviço, para as categorias de serviço (e parâmetros associados) relacionadas diretamente com a capacidade de operação de cada recurso real envolvido. O mecanismo de controle de admissão associado a cada recurso deve, então, ser acionado, a fim de verificar a viabilidade de aceitação do novo fluxo, utilizando-se das estratégias de admissão de recursos virtuais em cada um dos escalonadores escolhidos. Se todos os controladores de admissão responderem de forma afirmativa, os mecanismos de criação de recursos virtuais são acionados. Caso contrário, a requisição pode ser imediatamente negada, ou a negociação pode ser reiniciada, redistribuindo-se as parcelas de responsabilidade.

Durante a fase de manutenção de um contrato de serviço, ajustes sobre o sistema podem ser necessários, para que sejam asseguradas as especificações de QoS já requisitadas. A monitoração dos recursos reais visa a identificação de disfunções operacionais, seja por parte do usuário (e.g. fluxos submetidos fora da caracterização do tráfego), seja por parte do sistema (e.g. falha nos recursos, erros no cálculo das reservas). Os monitores devem emitir alertas ao mecanismo de sintonização na presença de algum distúrbio. As ações de sintonização podem envolver desde pequenos ajustes de parâmetros em determinados escalonadores, até a solicitação de uma renegociação geral da QoS.

2.4. Framework para adaptação de serviços

Embora os *frameworks* genéricos para provisão de QoS ofereçam a projetistas o conceito de pontos de flexibilização (*hot-spots*) específicos de serviço, permitindo a modelagem de sistemas adaptáveis, esses pontos apresentam relações indiretas de dependência entre si que dificultam a manutenção da consistência do sistema face a adaptações. Nesse contexto, a implementação de "meta-mecanismos", que automatizem a adaptação do sistema a novos serviços ou a novas políticas de provisão de QoS, e que observem questões como manutenção de consistência e restrições de reconfiguração relacionadas a segurança, é altamente desejável. O *framework* para adaptação de serviços foi elaborado neste trabalho para preencher parte dessa lacuna deixada pelos *frameworks* genéricos para provisão de QoS, tendo, no entanto, uma abordagem específica para sistemas operacionais.

As ações de adaptação requeridas pelos administradores do sistema, ou por um meta-mecanismo externo (que, no caso de redes programáveis por exemplo, pode ser um protocolo de sinalização aberto [Lazar97] ou outro mecanismo de gerência), devem ser controladas por um gerente de adaptação, responsável por receber as requisições, fazer verificações sobre a possibilidade de aceitação, inserir ou substituir o componente alvo e, finalmente, atualizar as referências nos mecanismos a ele relacionados. Para a criação de um serviço inteiramente novo, todos os componentes que implementam as políticas de provisão de QoS devem ser fornecidos ao gerente, juntamente com a localização da nova categoria na hierarquia de categorias de serviço.

Parte dos testes a que se refere o parágrafo anterior compreende a verificação de segurança da inserção do componente, que é delegada pelo gerente de adaptação a um agente específico. De um modo geral, o agente de verificação de segurança deve analisar cada novo componente levando em conta os seguintes aspectos básicos:

- Confiabilidade. O fornecedor do componente deve ser confiável.
- Restrição de contexto. As ações descritas pelo componente devem estar restritas ao contexto no qual o componente será aplicado.
- Isolamento. As ações descritas pelo componente, se logicamente erradas, não podem prejudicar a provisão de serviços para outras categorias ou a operação de outros subsistemas.

Se as verificações foram bem sucedidas, o gerente de adaptação submete a implementação do componente à *porta de adaptação* a ela correspondente. Portas de adaptação são as estruturas existentes no sistema operacional responsáveis por disponibilizar a implementação do componente aos mecanismos que a utilizarão. Um exemplo de porta de adaptação é o subsistema de módulos de *kernel* do *Linux*, apesar de

não ter sido criado exatamente para este fim (veja Seção 3). Finalmente, o gerente atualiza as estruturas que devem fazer referência ao novo componente, como um escalonador faz a um componente de criação ou a uma estratégia de escalonamento.

Um outro detalhe importante a ser observado está na remoção de componentes do sistema. Além da verificação de segurança, que confirma se o solicitante está autorizado para a ação, um outro teste, chamado de *verificação de consistência*, deve ser executado. O teste de consistência da remoção tem a responsabilidade de verificar se a remoção de um componente não acarretará o mau funcionamento ou total parada de outros componentes. Uma estrutura de dependências deve, então, ser mantida.

Nota-se que a funcionalidade dos mecanismos de adaptação pode ser aplicada não somente à infra-estrutura de provisão de qualidade de serviço, como também para outras partes do sistema operacional, como o subsistema de rede (pilha de protocolos), o gerenciamento de drivers, o sistema de arquivos, entre muitos outros. Obviamente, essa capacidade deve ser provida pelo *kernel*, atribuindo a esses subsistemas o suporte a portas de adaptação. Neste artigo são apresentados (na Seção 3) mecanismos de adaptação para a introdução de estratégias de admissão e de escalonamento nos buffers de comunicação do sistema *Linux*.

3. QoS no Subsistema de Rede do Linux

Com o objetivo de demonstrar como a arquitetura QoSOS pode ser aplicada na modelagem de um cenário real de provisão de QoS, esta Seção descreve a implementação do suporte aos serviços integrados (Intserv [Braden94]) sobre o subsistema de enfileiramento de pacotes de rede do sistema *Linux*. Para que fosse possível a instanciação do *framework* de adaptação de serviços e, em conseqüência, que estratégias de admissão e de escalonamento referentes às categorias de serviço Intserv pudessem ser substituídas em tempo de operação, o *kernel* do sistema teve de ser ligeiramente modificado. As modificações, no entanto, não serão aqui descritas por economia de espaço, podendo ser encontradas em [Moreno02].

3.1 – O Controle de Tráfego do Linux (LinuxTC)

As versões mais recentes de *kernel* do *Linux* oferecem um grande conjunto de funções de controle de tráfego de rede [Almesberger99]. As estruturas utilizadas para isso são capazes de oferecer os mecanismos de escalonamento necessários para o suporte às arquiteturas IntServ [Braden94] e DiffServ [Blake98]. O controle é composto pelos seguintes componentes conceituais: *disciplinas de enfileiramento*, *classes* e *filtros de classificação e policiamento*. Cada interface de rede tem a ela associada uma disciplina de enfileiramento, responsável por determinar como é tratado o enfileiramento neste dispositivo. Uma disciplina de enfileiramento pode ser simples tal qual aquela constituída apenas de uma única fila, como também pode ser mais elaborada e complexa, utilizando filtros para distinção dos pacotes, distribuindo-os entre classes.

O *LinuxTC* permite decidir a forma com que os pacotes devem ser enfileirados e quando eles devem ser descartados, por exemplo em uma situação em que o tráfego excede um certo limite. É possível também definir a ordem de envio desses pacotes, aplicando-se prioridades aos fluxos e, por último, retardar o envio de alguns pacotes para limitar a taxa de dados do tráfego de saída. Com este conjunto de ferramentas, é possível

configurar vários tipos de políticas para o escalonamento dos pacotes, distribuídas em uma estrutura hierárquica. Cada classe de uma disciplina de enfileiramento pode possuir uma nova disciplina para a qual será delegado o escalonamento de pacotes pertencentes àquela classe. O conceito de árvore de recursos virtuais, introduzido na Seção anterior, é genérico o suficiente para abranger também a utilização do controle de tráfego do *Linux* na instanciação da arquitetura proposta por este trabalho.

Por intermédio do programa `tc`, o *LinuxTC* oferece métodos muito ricos, mas pouco dinâmicos, para a configuração dos seus componentes. Isso porque os desenvolvedores desse subsistema não projetaram uma interface de programação para que aplicações e protocolos de negociação pudessem configurar as características de desempenho da comunicação pela rede. Recentemente, um projeto internacional de código aberto chamado TCAPI [Olshefski01] foi criado pela IBM com o objetivo de preencher essa lacuna. Escrita em linguagem C, a versão 1.2 da interface TCAPI foi utilizada para a configuração do *LinuxTC* no cenário proposto. Contudo, essa ferramenta mostrou-se deficiente, por não disponibilizar muitas operações importantes oferecidas pelo *LinuxTC* e, ainda, por conter algumas falhas de programação. Devido à facilidade de acesso ao código da interface TCAPI, várias modificações puderam ser feitas até que o funcionamento ideal, com as funções necessárias, fosse atingido. Tais alterações foram submetidas à apreciação do administrador do projeto e todas foram aprovadas e incorporadas ao software original, sendo uma das contribuições deste trabalho[3].

3.2 – Instanciação da Arquitetura QoSOS

A arquitetura instanciada será descrita utilizando uma abordagem orientada a objetos, em notação que segue a Linguagem de Modelagem Unificada [Rational97]. Adotou-se a convenção de que as classes-base da arquitetura apresentam-se preenchidas com a cor cinza. Quando a hierarquia de derivação de uma classe-base não for ilustrada, será utilizado um adorno do tipo "`<<classe-base>>`" sobre o nome da classe final, para indicar que se trata de uma especialização daquela classe-base. Na descrição textual, as classes e métodos abstratos estão notados de forma distinta, em *`itálico`*.

A instanciação da arquitetura QoSOS sobre o cenário de uso proposto levou ao desenvolvimento de duas interfaces de programação para aplicações (APIs) em Java, por meio das quais foram modelados muitos dos mecanismos descritos. A primeira API permite a solicitação de serviços com QoS por parte dos agentes de negociação de rede, promovendo controle de admissão e criação de recursos virtuais sobre o subsistema de enfileiramento de pacotes de rede. A segunda API permite que ações de adaptação sejam requisitadas pelos administradores de sistemas especificamente sobre as categorias de serviço acessíveis pela interface de solicitação de serviços.

A Figura 3 permite uma visão geral dessa instanciação, ilustrando como os componentes da API de solicitação interagem e modificam a estrutura do controle de tráfego, ações essas demarcadas por linhas tracejadas. A API de adaptação, por sua vez, tem seu fluxo de controle caracterizado pelas linhas pontilhadas, agindo apenas sobre a estrutura adaptável de estratégias de admissão e de escalonamento. O fluxo dos pacotes de rede

[3]TCAPI está disponível em http://www-124.ibm.com/developerworks/projects/tcapi/. O primeiro autor deste artigo mantém-se como colaborador do projeto.

entre as filas do controle de tráfego está representado pelas linhas contínuas. A descrição sucinta de cada um dos *frameworks* instanciados, a seguir, explica de forma completa a Figura 3. Maiores detalhes sobre as diversas classes definidas nos *frameworks* podem ser obtidos em [Moreno02].

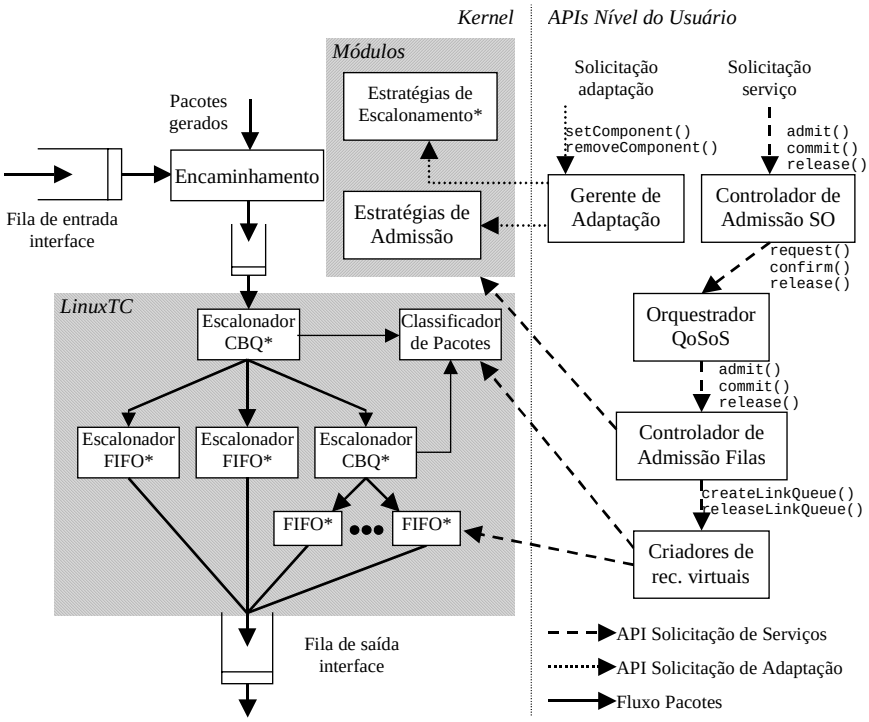

Figura 3 – Visão geral da implementação

Framework de parametrização de serviços

A Figura 4 ilustra o *framework* de parametrização de recursos. As categorias de serviço garantido e de carga controlada, definidas pelo modelo Intserv, são representadas pelas classes `GuarServiceCategory` e `CLServiceCategory`, respectivamente.

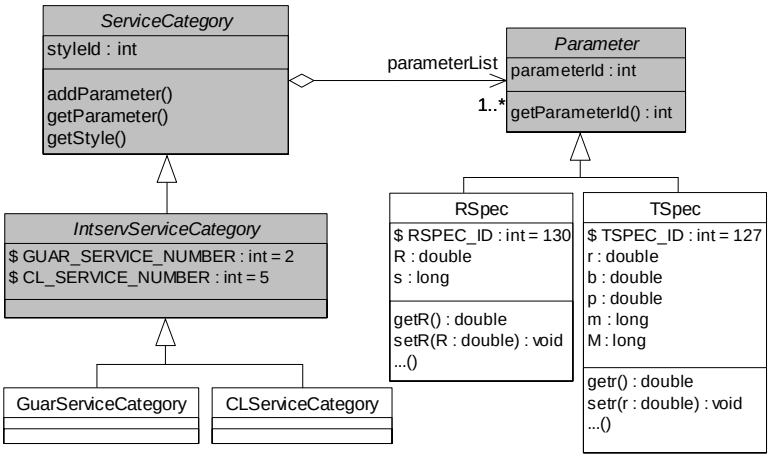

Figura 4 – Instanciação do framework de parametrização de serviços

O parâmetro `RSpec` (reservation specification) deve ser adicionado apenas a objetos da classe `GuarServiceCategory` por denotar os requisitos de qualidade a serem

reservados pelo sistema. Já o parâmetro `TSpec` (traffic specification) é de comum utilização por ambas categorias e descreve a caracterização do tráfego gerado pelo o usuário. R, s, r, b, p, m e M têm as definições dos parâmetros de mesmo nome do modelo IntServ [Braden94].

Não há a necessidade de definição de outros parâmetros relacionados com o recurso a ser alocado, já que o algoritmo CBQ (class based queue [Floyd95]) e os policiadores do *LinuxTC* são configurados por valores de mesmo significado que aqueles definidos pelas estruturas RSpec e TSpec.

Frameworks de compartilhamento de recursos

Os mecanismos do controle de tráfego do *Linux* puderam ser diretamente modelados pelo *framework* de escalonamento de recursos. A instanciação da arquitetura ilustrada pela Figura 5 reflete, inclusive, os nomes dos métodos encontrados internamente no *LinuxTC*, na forma como foram definidos em [Almesberger99]. A árvore de recursos virtuais inicial para o cenário proposto pode ter aspecto semelhante à árvore da Figura 2.

A classe `DeviceQueuingDiscipline` representa o escalonador raiz associado a uma interface de rede e responsável por receber os comandos `wakeup()` para iniciar a seqüência de escalonamento de pacotes (função `dequeue()`). Todas as outras disciplinas relacionadas com a mesma interface são modeladas pela classe `InnerQueuingDiscipline` e também utilizam a função `dequeue()` para dar prosseguimento ao escalonamento.

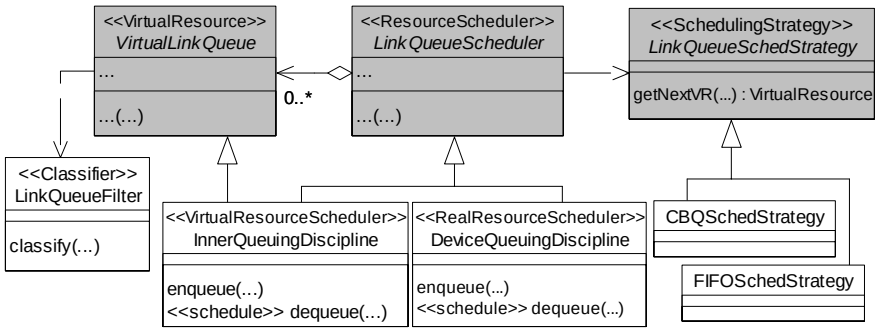

Figura 5 – Instanciação do framework de escalonamento de recursos

Quando um pacote é totalmente processado pela pilha de protocolos de rede, ele é colocado à disposição do escalonador raiz associado à interface de saída selecionada pelos procedimentos de encaminhamento. A seguir, essa disciplina raiz deve submeter o pacote à análise dos filtros classificadores, modelados por `LinkQueueFilter`, que identificam a categoria de serviço a que pertence o pacote. O escalonador raiz, então, invoca o método `enqueue()` do escalonador de recurso virtual correspondente.As estratégias de escalonamento oferecidas pelo *LinuxTC* que são utilizadas na provisão de serviços integrados estão representadas na modelagem pelas classes `CBQSchedStrategy` e `FIFOSchedStrategy`.

Entre os mecanismos modelados pela instanciação do *framework* de alocação de recursos, ilustrada na Figura 6, os componentes de criação de recursos virtuais não estão presentes no *LinuxTC* e precisaram ser implementados no espaço do usuário, como parte da API de solicitação de serviços. Foram instanciados dois componentes de criação, de

modo a corresponderem, cada um, a uma categoria de serviço específica, já que as necessidades de reserva entre elas são diferentes.

A alocação de recursos para os fluxos que requerem a categoria de serviço garantido é realizada por um objeto da classe `GuaranteedLQFactory`, por intermédio do método `createGuarLink()`. Essa operação pode ser dividida em duas etapas: a criação dos filtros de classificação e policiamento, e a reserva efetiva da largura de banda solicitada. Um único elemento "filtro" do *LinuxTC* realiza tanto o casamento de sua regra com os dados contidos no cabeçalho, como o policiamento dos pacotes casados seguindo sua regra de perfil de tráfego. Assim, o componente de criação da categoria de serviço garantido solicita a criação de um filtro de classificação e policiamento (através da interface TCAPI, pois o objeto está definido no espaço do usuário) invocando a função `change()` de um objeto da classe `LinkQueueFilter`. A reserva da largura de banda é feita pela criação de mais uma "classe" junto à disciplina de enfileiramento encarregada de escalonar os pacotes da categoria de serviço garantido. Isso é feito (também através da TCAPI) pela função `change()` pertencente à classe *`LinkQueueScheduler`*.

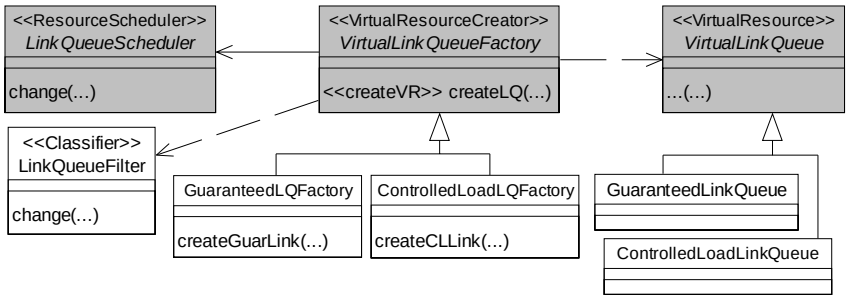

Figura 6 – Instanciação do framework de alocação de recursos

Com relação ao componente de criação de recursos virtuais associado à categoria de serviços de carga controlada, é suficiente apenas que o filtro seja configurado, pois não é definido pelo modelo Intserv a reserva de largura de banda para fluxos dessa categoria.

Frameworks de orquestração de recursos

Já completamente fora do ambiente do *LinuxTC*, a Figura 7 ilustra a instanciação do *framework* de negociação de QoS para o cenário proposto. Para atender uma solicitação de serviço feita por meio da primitiva `admit()`, o controlador de admissão do sistema operacional (classe `QoSoSAdmissionController`) encaminha os parâmetros fornecidos ao orquestrador de recursos (classe `QoSoSOrchestrator`). O orquestrador deve repassar os parâmetros solicitados ao controlador de admissão das filas de enlace (classe `LinkQueuingAdmissionController`, por meio do método `admit()`). Isso pode ser feito diretamente, como dito anteriormente.

O controlador de admissão das filas verifica a categoria de serviço solicitada e invoca o método `check()` da estratégia de admissão correspondente (classe *`LinkQueueAdmissionStrategy`*). Esse método analisa os atuais parâmetros de desempenho dos recursos e os compara com a necessidade de QoS solicitada. Se ficar concluído que a solicitação é viável, o controlador de admissão é informado e gera um identificador de pré-reserva, que será utilizado posteriormente na confirmação do serviço. O controlador de admissão das filas retorna o identificador gerado ao orquestrador QoSOS, que repassa ao controlador de admissão QoSOS.

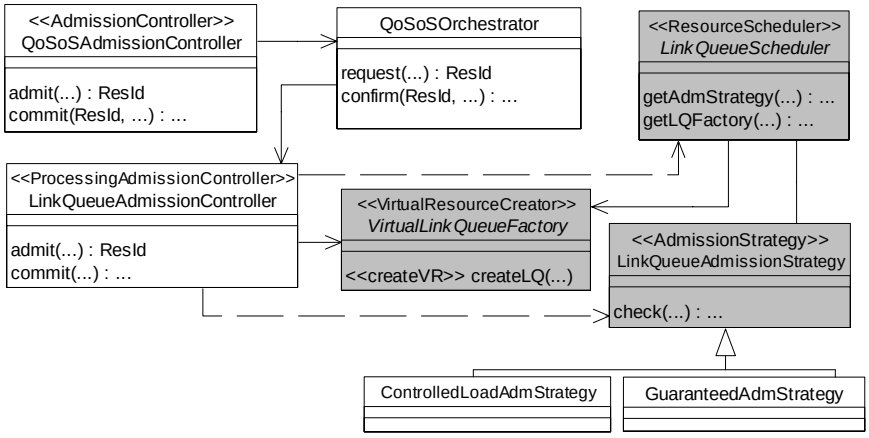

Figura 7 – Instanciação do framework de negociação de QoS

Com o objetivo de confirmar o serviço, deve ser invocado o método `confirm()` do orquestrador QoSOS, informando o identificador da pré-reserva, que é repassado pelo orquestrador ao controlador de admissão das filas, utilizando o método `commit()`. Se o identificador ainda for válido, os dados da pré-reserva são recuperados e o componente apropriado de criação de recursos virtuais (uma especialização da classe *`LinkQueueFactory`*) é acionado, como descrito no *framework* de alocação de recursos. Essa seqüência de chamadas é ilustrada resumidamente pela Figura 3.

As estratégias de admissão disponíveis inicialmente correspondem às classes `GuaranteedAdmStrategy` e `ControlledLoadAdmStrategy`. Para que a admissão de um fluxo de serviço garantido seja aprovada, a taxa de dados (R) deve estar disponível no escalonador de recurso virtual. Essa estratégia é conhecida como *soma simples* e corresponde à estratégia A, especificada na estrutura da árvore de recursos virtuais, apresentada pela Figura 2. Já a admissão para carga controlada é regida pela estratégia B da figura, que consiste em testar se a soma dos parâmetros r de todos os fluxos anteriormente admitidos mais o parâmetro r do fluxo que solicitou o serviço não excede a largura de banda alocada à categoria de serviço. Essa estratégia é equivalente à *soma simples*, mas leva em conta os parâmetros de caracterização do tráfego.

Framework de adaptação de serviços

O *kernel* monolítico do sistema operacional *Linux* possui um subsistema de gerência de módulos de *kernel*, que inclui funções de inserção, remoção, verificação da necessidade e teste de utilização para os módulos. Esse subsistema foi projetado inicialmente para a implementação de drivers de dispositivos e a conseqüente redução do tamanho do *kernel*. Vários trabalhos na área de sistemas operacionais utilizaram essa funcionalidade para realizar a configuração de partes internas do *kernel*, como o escalonamento de processos [Barabanov97]. Nota-se, entretanto, que por não ser a adaptabilidade em tempo de operação uma filosofia de projeto do *Linux*, várias são as modificações que devem ser feitas no próprio *kernel* para que ele aceite e utilize uma nova estratégia de escalonamento de pacotes, por exemplo. Por economia de espaço, essas modificações não serão discutidas neste artigo, podendo ser encontradas em [Moreno02].

A Figura 8 ilustra a instanciação do *framework* de adaptação de serviços sobre a infra-estrutura desenvolvida dentro do *kernel*, com as modificações anteriormente

mencionadas. A classe `LinuxQoSAdaptationManager` representa o gerente de adaptação, implementado no espaço do usuário. Sua funcionalidade está restrita à inserção e substituição de módulos que implementam estratégias de admissão e estratégias de escalonamento de pacotes. Tais módulos são abstraídos sob a forma de componentes adaptáveis, através das classes `AdmissionStrategyComponent`, `SchedulingStrategyComponent`, `KernelModulePort` e `ObjectFile`. Nota-se que, sendo possível a adaptação das estratégias de escalonamento, o administrador do sistema não fica limitado apenas aos algoritmos disponibilizados pelo *LinuxTC*.

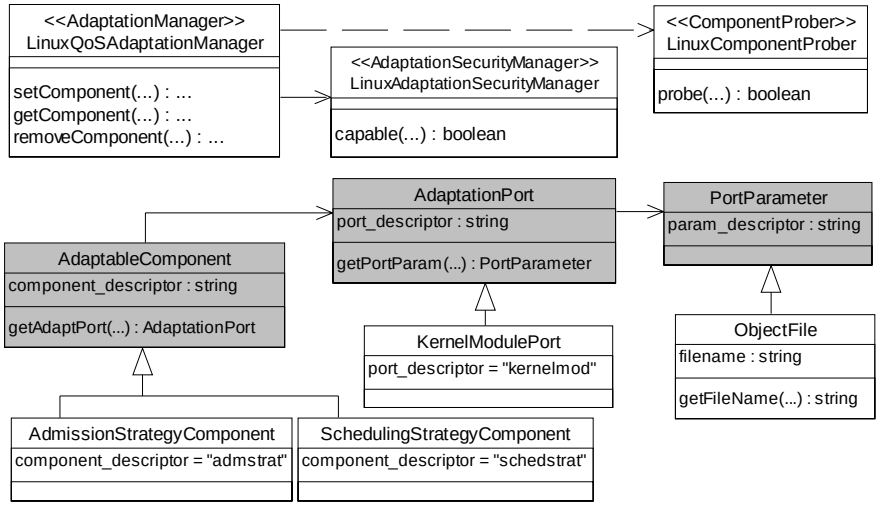

Figura 8 – Instanciação do framework de adaptação de serviços

Quando o administrador do sistema, ou qualquer outro mecanismo de gerência autorizado, solicita a inserção ou substituição (`setComponent()`) de um componente, o gerente de adaptação instanciado se certifica de que a porta de adaptação corresponde ao subsistema de módulos do *kernel*, para o qual será submetida a implementação do componente por meio da chamada `ins_mod`. Esse subsistema possui uma verificação simples de segurança, modelada pela classe `LinuxAdaptationSecurityManager`, que consiste na autenticação das capacidades atribuídas ao solicitante, seja ele um usuário ou um programa (função `capable()`).

4. Trabalhos Relacionados

Os trabalhos relacionados à provisão de QoS em sistemas operacionais incluem desde propostas de extensão para GPOS a projetos de novos sistemas operacionais, todos com foco sobre mecanismos de gerenciamento de recursos.

As referências [Goyal96], [Ford96] e [Regher01] propõem novas estruturas para escalonamento hierárquico de processos, para permitir justiça no escalonamento frente às diferentes necessidades das aplicações. Isso é feito por meio do uso de diferentes escalonadores de processos para cada classe de aplicação definida. Tais arquiteturas prevêem a possibilidade de adaptação do código dos escalonadores em tempo de execução. Em nosso trabalho, a estrutura de escalonamento hierárquico é estendida para manipular outros tipos de recursos, como já demonstrado, a partir da generalização oferecida pela árvore de recursos virtuais, uniformizando a modelagem.

O algoritmo LDS (Load Dependent Scheduler) [Barria00] é capaz de emular a operação de vários algoritmos de escalonamento, a partir da modificação dos seus próprios parâmetros de operação. Mesmo concebido com o objetivo de se tornar uma ferramenta para análise e comparação do desempenho de algoritmos de escalonamento, o LDS é também uma ótima solução para adaptação, pois a inserção de um novo escalonador pode ser feita sem a necessidade de programação. A árvore de recursos virtuais e o framework de adaptação de serviços são genéricos o suficiente para abranger o LDS.

Várias propostas de arquiteturas para o subsistema de rede de sistemas operacionais tentam solucionar outros problemas citados neste artigo. Sistemas como Sumo [Coulson95] e Nemesis [Black97] foram projetados para oferecer suporte a aplicações multimídia distribuídas, sendo baseados na arquitetura de *microkernel*. Uma de suas características principais é a definição da pilha de protocolos no espaço do usuário. Cada aplicação deve ter sua própria instanciação da pilha, com buffers dedicados aos seus fluxos. Já a arquitetura LRP (*Lazy Receiver Processing*) [Druschel96] é uma extensão ao padrão NetBSD, com o objetivo de diminuir o número de interrupções no processamento de recepção da pilha de protocolos. Nota-se que, para os vários tipos de implementação do subsistema de rede, diferentes estratégias de orquestração de recursos podem ser necessárias, característica esta contemplada pelo framework de negociação apresentado neste artigo.

5. Conclusões e Trabalhos Futuros

Neste trabalho foi apresentada a arquitetura QoSOS para provisão de QoS adaptável em sistemas operacionais. A arquitetura proposta é resultado da especialização e instanciação de *frameworks* genéricos de QoS que definem estruturas recorrentes e uma representação homogênea para os mecanismos de QoS e interfaces de serviços.

Protótipos implementados utilizando o sistema *Linux* foram expostos para descrever como a arquitetura pode ser instanciada, demonstrando como um GPOS pode prover mecanismos de provisão de QoS a despeito de todas as deficiências que se apresentam na implementação de tal tipo de serviço. Esses esforços de implementação resultaram em contribuições para o projeto internacional de código aberto TCAPI, além de integrar um projeto para a construção de uma infra-estrutura de suporte a aplicações de TV interativa. Entre as metas inclui-se a extensão do sistema *Linux* para a provisão de QoS.

O presente trabalho abriu possibilidades de aprofundamento dos seguintes assuntos, passíveis de exploração em trabalhos futuros:

- Inclusão de outros recursos relevantes, como memória, sistema de paginação e acesso a disco no modelo de orquestração.
- Implementação de mecanismos de QoS no escalonamento de processos do sistema *Linux*: Os mecanismos de QoS para escalonamento de processos encontram-se modelados para a próxima instanciação da arquitetura, incluindo a possibilidade de utilização do algoritmo LDS, tanto nesse contexto quanto no *LinuxTC*. Com o gerenciamento de QoS também sobre a CPU, passa a ser interessante uma avaliação do desempenho do sistema, para a validação do modelo de orquestração.
- Refinamento do framework de adaptação, incluindo procedimentos para a verificação das questões de segurança e de consistência nas ações de inclusão e de substituição de componentes.

Referências

Almesberger, W. (1999) “Linux network traffic control: Implementation overview”. Implementation Details <ftp://icaftp.epfl.ch/pub/people/almesber/pub/tcio-current.ps.gz>.

Barabanov, M. (1997) “A linux-based real-time operating system”. Master Degree dissertation, New Mexico Institute of Mining Technology, 1997.

Barria, M., Vallejos, R., Soares, L. F. G. (2000) “LDS: A Load Dependent Scheduling Algorithm”. IFIP ATM & IP 2000 Workshop, In Participants Proceedings.

Black, R., et al. (1997) “Protocol implementation in a vertically structured operating system”. Proceedings of 22nd Conference on Local Computer Networks (LCN’97).

Blake, S. et al. (1998) “An architecture for differentiated services”. IETF RFC2475.

Braden, R.; Clark, D.; Shenker, S. (1994) “Integrated services in the internet architecture: an overview”. IETF RFC1633.

Campbell, A. et al. (1999) “A Survey of Programmable Networks”. ACM SIGCOMM Computer Communications Review, v.29, n.2, p. 7-23.

Coulson, G., Blair, G. (1995) “Architectural principles and techniques for distributed multimedia application support in operating systems”. ACM Operating Systems Review, v.29, n.4, p.17-24.

Druschel, P., Banga, G. (1996) “Lazy receiver processing (LRP): a network subsystem architecture for server systems”. Proceedings of 2nd Symposium on Operating System Design and Implementation (OSDI'96), p. 261-275.

Floyd, S.; Jacobson, V. (1995) “Link-sharing and Resource Management Models for Packet Networks”. IEEE/ACM Transactions on Networking, v.3, n.4, p. 365-386.

Ford, B., Susarla, S. (1996) “CPU inheritance scheduling”. Proceedings of 2nd Symposium on Operating Systems Design and Implementation (OSDI'96).

Gomes, A.T.A., Colcher S., Soares, L.F.G. (2001) “Modeling QoS Provision on Adaptable Communication Environments”, Proceedings of the IEEE International Communication Conference (ICC2001), Helsinki, Finland.

Goyal, P., Guo, X., Vin, H. (1996) “A hierarchical CPU scheduler for multimedia operating systems”. Proceedings of 2nd Symposium on Operating System Design and Implementation (OSDI'96), p. 107-122.

Kosmas, N., Turner K. (1997) “Requirements for service creation environments”. 2nd International Workshop on Applied Formal Methods in System Design, p. 133-137.

Lazar, A. (1997) “Programming telecommunication networks”. IEEE Network Magazine, no. 5, p. 8-18.

Moreno, M.F. (2002) “Um framework para provisão de QoS em sistemas operacionais”. Dissertação de Mestrado, Pontifícia Universidade Católica do Rio de Janeiro.

Olshefski, D. (2001) “Notes on linux network QoS – TCAPI version 1.0”. Work in progress. <ftp://www-126.ibm.com/pub/tcapi/tcapi.tar.gz>

Rational Software Corporation. (1997) “Unified Modeling Language: Notation Guide”.

Regher, J. (2001) “Using Hierarchical Scheduling to Support Soft Real-Time Applications in General-Purpose Operating Systems”. PhD thesis, Univ. of Virginia.