Árvores de Recursos Virtuais: Um modelo de gerenciamento de recursos fim-a-fim com QoS

Marcelo F. Moreno, Sérgio Colcher, Luiz Fernando Gomes Soares

Depto. de Informática – Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio)
Rua Marquês de São Vicente, 225 RDC – 22453-900 – Rio de Janeiro - Brasil

{moreno, colcher,lfgs}@inf.puc-rio.br

Abstract. *This work proposes an end-to-end resource management model that allows the coexistence of different management policies over each resource and offers adaptation mechanisms that make possible the fast deployment of new services in face of the continuous evolution of distributed applications. We also present a framework for instantiation of the model as an extension to operating systems in general.*

Resumo. *Este trabalho propõe um modelo de gerenciamento de recursos fim-a-fim que permite a coexistência de diferentes políticas de gerenciamento sobre cada recurso e oferece mecanismos de adaptação que possibilitam a rápida implantação de novos serviços frente à contínua evolução das aplicações distribuídas. Demonstra-se, também, um framework para a instanciação do modelo como uma extensão a sistemas operacionais em geral.*

1. Introdução

O uso de aplicações multimídia distribuídas alcançou um nível de disseminação tal que, hoje em dia, não se pode ignorar a necessidade de atendimento aos seus requisitos de desempenho, de modo fim-a-fim. Essa funcionalidade é alcançada por meio de mecanismos de provisão de qualidade de serviço (*QoS – Quality of Service*), aplicados em todos os subsistemas de gerenciamento de recursos participantes.

Presentes em todos os nós de um sistema distribuído (servidores, estações finais e *gateways*), os sistemas operacionais representam pontos centrais de gerenciamento de recursos e, portanto, possuem um papel fundamental na provisão de QoS. Inevitavelmente, tais recursos diferem em diversos aspectos, como natureza, finalidade, tecnologia e mídia. A complexidade da provisão de QoS fim-a-fim sobre tal cenário heterogêneo pode ser amenizada se os mecanismos de gerenciamento forem manipulados uniformemente, desde sua implementação nos sistemas operacionais.

A provisão de QoS é ainda mais complexa quando se considera a contínua evolução das aplicações, que resulta no surgimento recorrente de novos requisitos. Por isso, são também habilidades essenciais para os sistemas operacionais: i) a capacidade de manipulação concomitante de diferentes políticas de provisão de QoS sobre um mesmo recurso e ii) a flexibilidade (ou adaptabilidade) para a implantação rápida e dinâmica de novos serviços com QoS.

O objetivo deste trabalho é propor um modelo de gerenciamento de recursos fim-a-fim com QoS, que possa ser instanciado como extensão a sistemas operacionais, para um controle direto e de alta granularidade sobre os recursos. O modelo, denominado *Árvores de Recursos Virtuais* (*Virtual Resource Trees – VRT), se baseia em uma abstração única para a uniformização dos mecanismos de gerenciamento, capaz de representar os diferentes tipos de recursos individualmente, bem como a composição*

deles em um cenário fim-a-fim. Tal abstração contempla os requisitos de adaptabilidade e coexistência de diferentes *políticas de gerenciamento* sobre um mesmo recurso.

VRT representa a base de uma ampla arquitetura de gerenciamento de recursos em desenvolvimento, chamada DHARMA (*Distributed, Homogeneous and Adaptable Resource Management Architecture*). DHARMA inclui ferramentas e linguagens que facilitam o projeto e implantação de vários aspectos inerentes ao problema. Neste artigo, no entanto, o escopo fica restrito ao modelo VRT e técnicas para sua instanciação.

2. O Modelo de Árvores de Recursos Virtuais

Árvore de recursos virtuais (VRT) é uma abstração que denota a *divisão hierárquica* da capacidade de um recurso ou de um conjunto de recursos gerenciados em grupo. O nó raiz representa a totalidade do recurso (ou de um grupo de recursos) a ser compartilhado hierarquicamente quanto à sua *alocação* (admissão/reserva) e *escalonamento*. Cada nó da árvore é chamado *recurso virtual* e representa uma porção da capacidade do seu nó pai. A Figura 1 ilustra uma VRT genérica.

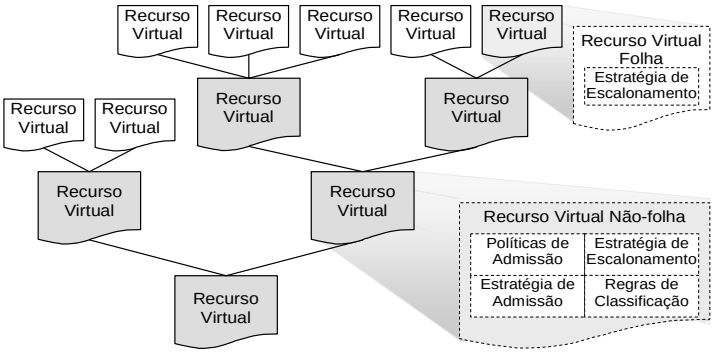

Figura 1 - VRT genérica

Um *escalonador* é associado a cada recurso virtual, distribuindo sua capacidade entre seus nós filhos. O significado exato de "capacidade" varia de acordo com o recurso. Por exemplo, para o recurso CPU, trata-se da capacidade de processamento, expressa em termos de tempo de uso, vazão, taxa média, etc. O escalonador associado à raiz da árvore compartilha a capacidade total do recurso entre os recursos virtuais filhos. Os demais escalonadores escolhem qual entre seus nós filhos fará uso de sua capacidade alocada junto ao nó pai. Na realidade, a tarefa de escolha realizada pelos escalonadores se baseia em algoritmos encapsulados em *estratégias de escalonamento*.

Normalmente, recursos podem ser escalonados no tempo ou no espaço, sendo que é possível um mesmo recurso ser escalonado em ambas as dimensões. Por isso, VRTs podem denotar tanto o compartilhamento temporal quanto o espacial. No caso em que um mesmo recurso é escalonado em ambas as dimensões, duas VRTs distintas devem ser associadas a ele. Um exemplo desse tipo de recurso é o disco, que é escalonado no tempo, em sua fila de acesso, e no espaço, em sua área de armazenamento.

Da raiz da árvore até seus nós folhas, a capacidade do recurso é dividida hierarquicamente e finalmente alocada para fluxos (de dados, instruções, etc.). Quando unidades de informação de um fluxo chegam à raiz de uma VRT (e.g. pacotes de rede em uma interface), um classificador associado à VRT as analisa para determinar o recurso virtual folha onde serão escalonadas. Ele se baseia em *regras de classificação* configuradas na raiz e em nós intermediários sempre que folhas são criadas. Cada regra

especifica (i) um padrão de casamento a ser testado contra o conteúdo das unidades de informação e (ii) um recurso virtual alvo, cujas regras de classificação serão as próximas testadas em caso de sucesso, recorrentemente, até que o recurso virtual alvo seja folha.

Exceto os nós folhas, cada recurso virtual possui associado um *controlador de admissão*, responsável por verificar se uma solicitação de criação de um recurso virtual é permitida e viável. Uma criação é permitida se suas características satisfazem a *políticas de admissão*, que formam um conjunto de regras restritivas que incidem sobre o recurso virtual pai dessa criação. Já a viabilidade da criação de um recurso virtual é verificada por meio de uma *estratégia de admissão*, que normalmente leva em conta as solicitações já admitidas e a solicitação em análise para inferir sobre tal viabilidade. Uma vez verificada ser possível e viável, a criação do recurso virtual pode ser efetivamente realizada pelo controlador de admissão, que configura os parâmetros de escalonamento do recurso virtual pai e cria as novas regras de classificação necessárias.

De uma forma geral, o conjunto de algoritmos e regras envolvidos na alocação e no escalonamento de um recurso é denominado "*políticas de gerenciamento*". As políticas de gerenciamento de uma VRT incluem as regras de classificação, estratégias de escalonamento, estratégias de admissão e políticas de admissão utilizadas por seus recursos virtuais. No modelo VRT, cada política é desacoplada dos escalonadores e controladores de admissão, de forma a oferecer pontos de flexibilização (*hot-spots*) que podem ser preenchidos em tempo de projeto. Em instanciações reconfiguráveis (extensíveis), esses *hot-spots* podem, ainda, ser modificados em tempo de operação, sem necessidade de interrupção de serviços não-relacionados à adaptação. A capacidade de manipulação de diferentes políticas de gerenciamento sobre o mesmo recurso e de modificação dessas políticas em tempo de operação permitem que as instâncias do modelo VRT atendam à contínua evolução dos requisitos das aplicações.

Árvores de recursos virtuais primitivas

No caso mais simples, uma VRT representa um único recurso real, como, por exemplo, quando sua raiz está associada a uma CPU ou um canal de comunicação. Alternativamente, a raiz de uma VRT pode representar um recurso virtual, filho de um recurso real, que foi separado da árvore por simplicidade de gerenciamento. Por exemplo, uma parcela de CPU pode se tornar a raiz de uma VRT dedicada a um usuário específico do sistema operacional. Uma raiz pode também representar um conjunto de recursos reais homogêneos, gerenciados como se fossem um único recurso, ,como em sistemas multiprocessados. Nesses três casos, a VRT é dita ser uma *VRT primitiva*, por ter suas políticas de gerenciamento atuando diretamente sobre recursos reais.

Árvores de recursos virtuais compostas e florestas

Recursos gerenciados independentemente podem ser alvos de uma orquestração [Moreno03], resultando na construção de uma VRT cuja raiz representa uma composição de diferentes recursos virtuais, situação em que a árvore é denominada uma *VRT composta* (Figura 2). No cenário de provisão de QoS fim-a-fim, vários recursos estarão envolvidos na formação de uma VRT composta, como por exemplo enlaces de rede, CPUs, discos, memória, etc. VRTs compostas possibilitam, então, que todo um caminho de recursos seja representado em algum nível de abstração, de forma que sua gerência possa ser vista como se atuasse sobre um único recurso. Uma VRT composta é o resultado da orquestração de recursos envolvendo um conjunto de VRTs que compõem uma *floresta de recursos* (e.g. sistemas operacionais, sub-redes, etc).

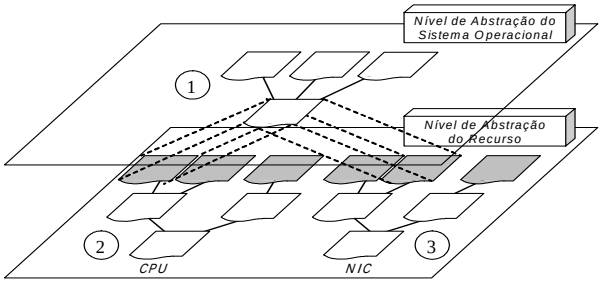

Figura 2 - Exemplo de VRT composta

3. VRT-FS: Um framework para instanciação do modelo VRT

A instanciação do modelo VRT demanda a criação de uma estrutura de dados complexa e flexível o suficiente para acomodar adaptações sobre políticas de gerenciamento. Dentre as soluções investigadas, o uso de sistemas de arquivos especiais como estrutura de dados e interface para o gerenciamento de recursos se mostrou uma solução de projeto [Bruno99] muito interessante: i) sistema de arquivos é uma abstração inerentemente hierárquica, tal qual o conceito de VRTs; ii) sistemas de arquivos especiais podem se tornar ainda mais atrativos ao oferecerem uma interface para a modificação do comportamento do sistema e iii) operações sobre sistemas de arquivos formam uma interface padronizada, facilmente mapeada para operações em VRTs.

A instanciação do modelo VRT baseada em um sistema de arquivos especial é chamada VRT-FS e sua especificação deve ser vista como um *framework*. De fato, VRT-FS descreve a estrutura de arquivos, diretórios e elos, e o comportamento esperado do sistema frente às interações com tais elementos. No entanto, detalhes sobre como a estrutura de arquivos deve ser integrada internamente aos subsistemas de gerenciamento de cada recurso são deixados em aberto, dado que são idiossincrasias de cada sistema.

A raiz de VRT-FS é o diretório `/vrt`, sob o qual se encontram outros três diretórios: `/vrt/primitive`, onde são representadas VRTs primitivas; `/vrt/forest`, que abriga florestas; e `/vrt/composite`, onde VRTs compostas são mantidas. Esta seção se concentra apenas na discussão sobre VRTs primitivas por limitações de espaço.

As VRTs primitivas de recursos gerenciados pelo sistema local são representadas sob o diretório `/vrt/primitive`, ilustrado pela Figura 3. Devido ao fato de que recursos podem ser escalonados no tempo ou no espaço, ele possui dois subdiretórios: `./temporal` e `./spatial`. Ambos possuem a mesma estrutura: seus diretórios filhos representam os recursos virtuais raízes das VRTs de cada recurso gerenciado pelo sistema e são criados automaticamente durante a inicialização de VRT-FS.

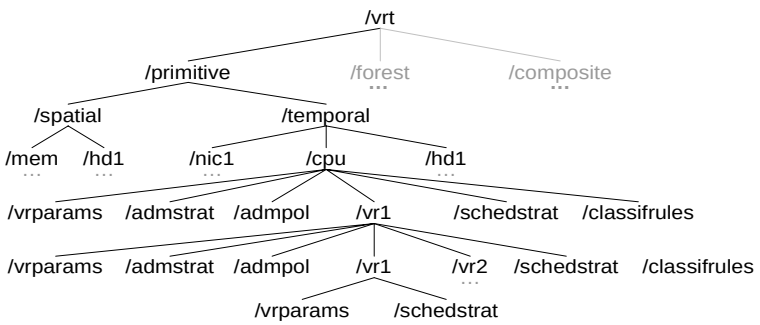

Figura 3 – VRT-FS: `/vrt/primitive`

Na Figura 3, o foco é dado a uma CPU VRT, sendo sua estrutura expandida como exemplo e para acompanhamento das próximas definições. Todos os subdiretórios sob uma raiz de VRT representam recursos virtuais filhos dessa raiz. Com a representação de recursos virtuais por meio de diretórios, o caminhamento entre recursos virtuais é trivialmente mapeado para instruções *chdir*. Na figura, `./vr1`, `./vr1/vr1` e `./vr1/vr2` são exemplos de diretórios de recursos virtuais filhos da raiz `cpu`.

Todo diretório de recurso virtual possui um arquivo chamado `vrparams`, que se dispõe a ser uma interface para a manutenção da alocação concedida ao recurso virtual. A instrução *read* sobre `vrparams` retorna a capacidade alocada ao recurso virtual representado pelo diretório que o contém. A instrução *write* altera a capacidade alocada, o que demanda um novo processo de admissão, detalhado mais a frente.

Diretórios de recursos virtuais podem possuir outros quatro arquivos, que permitem a manutenção de cada uma das políticas de gerenciamento associadas ao recurso virtual: `classifrules` - regras de classificação; `schedstrat` - estratégia de escalonamento; `admstrat` - estratégia de admissão e `admpol` - políticas de admissão. O conteúdo desses arquivos depende da forma pela qual a instanciação do modelo implementa tais políticas de gerenciamento nos sistemas operacionais envolvidos. Algumas possibilidades são: descrições de regras [Kotsovinos06], módulos de núcleo [Salzman05], serviços de micronúcleo e componentes de software. A instrução *read* sobre esses arquivos retorna o conteúdo do arquivo carregado, ao passo que a instrução *write* dispara a inclusão (ou substituição) da nova política de gerenciamento no sistema. Somente recursos virtuais não-folhas possuem os arquivos `admstrat`, `admpol` e `classifrules`, uma vez que recursos virtuais folhas não possuem um controlador de admissão e são nós finais em uma classificação.

Recursos virtuais são criados por meio da instrução *mkdir*, permitida apenas sob diretórios que possuem o arquivo `admstrat`. O nome do novo diretório é livre. O sistema deve disponibilizar automaticamente os arquivos `vrparams` (nulo) e `schedstrat` (com alguma estratégia padrão, como FIFO, round robin...) sob o novo diretório. Note que o processo de admissão ainda não se iniciou, pois ainda não foram passados os parâmetros de QoS desejados para o novo recurso virtual. Por isso, *read* sobre `vrparams` retorna nulo até que o processo de admissão seja bem sucedido. Os parâmetros de QoS são fornecidos como conteúdo de um *write* sobre `vrparams` e devem ser compatíveis com aqueles esperados pela estratégia de admissão do nó pai. Caso contrário, *write* retornará erro, como se a admissão fosse negada. A instrução *write* somente retornará com sucesso se o controlador de admissão aprovar a requisição.

Por convenção, todo novo recurso virtual é folha. Para que um recurso virtual se habilite a ser pai de outros, basta a criação do arquivo `admstrat` com alguma estratégia de admissão codificada no seu conteúdo (no mínimo, uma estratégia que sempre responda afirmativamente). O arquivo `admpol` será automaticamente criado, devendo ser *adapt*ados posteriormente.

Finalmente, vale a pena citar que os diretórios de VRTs compostas e florestas fazem uso intensivo de elos (*links*) para apontar os recursos virtuais folhas participantes de uma composição e as raízes que participam de cada floresta. Tais ponteiros podem ter como alvos diretórios locais e/ou remotos. Note que, diferentemente de VRTs primitivas, florestas e VRTs compostas são criadas por gerência ou por negociadores de

QoS [Moreno03]. Note também que os diretórios de VRTs compostas também possuem arquivos como `admstrat` e `admpol` para possibilitar a criação de recursos virtuais filhos.

4. Trabalhos Relacionados e Futuros

Os modelos de qualidade de serviço implementados em sistemas operacionais, tais como [Shenoy98][Almes99][Regher01][Lawall04], não abrangem o problema em sua totalidade, por incluírem apenas recursos específicos ou não considerarem de forma integrada os mecanismos de provisão de QoS e os de adaptação de serviços. Mesmo as soluções mais completas não definem um modelo de recurso fim-a-fim e que também norteie o desenvolvimento e adaptações das políticas de gerenciamento.

Quanto às arquiteturas de gerenciamento de recursos implementadas em *middleware*, nota-se que possuem forte preocupação com a sintonização de QoS [Blair01][Cardei00], uma vez que os recursos em si não são diretamente controlados e violações podem ocorrer freqüentemente. Ainda assim, nas propostas de *middleware* que se integram a sistemas operacionais específicos [Kon00][Bavier04], não há a flexibilização de diferentes políticas de compartilhamento dos recursos.

A arquitetura DHARMA, que se encontra em desenvolvimento, possui como objetivo principal oferecer um controle de alto grau sobre o gerenciamento de recursos. DHARMA prioriza a uniformização do gerenciamento (baseado no modelo VRT), a implementação dos mecanismos de adaptação (por meio de VRT-FS) e a construção de ferramentas de criação e modificação de serviços com QoS. Essas ferramentas formam um ambiente de alto nível para a (i) especificação de políticas de gerenciamento e (ii) o projeto e implantação distribuída de VRTs. Para cada um desses aspectos, linguagens de domínio específico (DSLs) [Lawall04] aproximam tais tarefas do domínio de conhecimento dos administradores.

5. Conclusão

O presente artigo descreveu um modelo de gerenciamento de recursos baseado no conceito de Árvores de Recursos Virtuais. A partir dessa abstração única, o modelo uniformiza os mecanismos de gerenciamento, permitindo a representação de diferentes tipos de recursos, assim como a composição deles em um cenário fim-a-fim. Tal abstração é, ainda, flexível o suficiente para contemplar os requisitos de adaptabilidade e de coexistência de diferentes políticas de gerenciamento sobre um mesmo recurso.

Um *framework* para instanciações de VRT também foi apresentado, baseado em um sistema de arquivos especial, denominado VRT-FS. VRT-FS captura todas as funcionalidades e características do modelo VRT e as mapeia sobre uma estrutura de arquivos, diretórios e elos, descrevendo o comportamento esperado do sistema frente às interações com tais elementos. O andamento de uma instanciação de VRT-FS sobre o núcleo Linux pode ser consultada em http://qososlinux.telemidia.puc-rio.br/vrtfs.html.

Referências

Almesberger, Werner. (1999) Linux network traffic control – Implementation overview. Disponível em http://snafu.freedom.org/linux2.2/docs/tcio-current.ps.

Bavier, A. et al. (2004). “Operating System Support for Planetary-Scale Network Services”. In: Proceedings of Networked Systems Design and Implementation 2004 (NSDI'04).

Blair, G. et al. (2001) The Design and Implementation of Open ORB version 2. IEEE Distributed Systems Online, v.2, n.6.

Bruno, J. et al. (1999). “Retrofitting Quality of Service into a Time-Sharing Operating System”. Proceedings of the 1999 USENIX Annual Technical Conference.

Cardei, I. (2000). Hierarchical Architecture for Real-Time Adaptive Resource Management. Proceedings of IFIP/ACM Middleware Conference.

Goyal, Pawan; Guo, Xingang; Vin, Harrick. (1996) A hierarchical CPU scheduler for multimedia operating systems. Proceedings of 2nd Symposium on Operating System Design and Implementation (OSDI'96), p. 107-122.

Kon, F. et al. (2000) 2K: A Distributed Operating System for Dynamic Heterogeneous Environments. IEEE International Symposium on High Performance Distributed Computing.

Kotsovinos, Evangelos; Ion, Iulia; Harris, Tim (2006). Resource Management for Global Public Computing: Many Policies Are Better Than (N)one. USENIX WORLDS 2006 Nov 2006

Lawall, J.; Muller, G.; Duchesne, H. (2004) Language Design for Implementing Process Scheduling Hierarchies. Symposium on Partial Evaluation and Program Manipulation.

Moreno, Marcelo et al (2003). “QoSOS: An adaptable architecture for QoS provisioning in network operating systems”. Journal of the Brazilian Telecommunications Society, Special Issue, v.18, n.2, p.118-131.

Regehr, John. (2001) Using Hierarchical Scheduling to Support Soft Real-Time Applications in General-Purpose Operating Systems. PhD thesis, University of Virginia.

Salzman, P.; Burian, M.; Pomerantz, O. (2005). The Linux Kernel Module Programming Guide. Página na Internet: http://www.tldp.org/LDP/lkmpg/2.6/html.

Shenoy, P.; Vin, H. Cello: A (1998) Disk Scheduling Framework for Next Generation Operating Systems", Proceedings of SIGMETRICS'98, ACM.