Otimização de Escalonamento no Xen

Juliano Potrich[1], **Fábio Diniz Rossi**[2]

[1]Conectt
Rua Uruguaiana, 10, Cj. 1801 Centro – CEP 20050.090 - RJ - Brazil

[2]Campus Alegrete
Instituto Federal Farroupilha
RS 377 - Km 27 - Caixa Postal 118 - Alegrete - RS - Brazil

juliano.potrich@conectt.com.br, fdrossi@al.iffarroupilha.edu.br

Abstract. *This paper shows an analysis of the Xen virtual machine monitor, describing Credit Scheduler internal structure. The main goal of this work is to change a Xen scheduler in order to improve the performance of the virtual machines that are running over Xen. We present the places in which it is possible to make some improvements in the Credit Scheduler. An analysis of the new Credit scheduler is presented and shows the improvements for the overall system.*

Resumo. *Este artigo apresenta uma análise do monitor de máquinas virtuais Xen, descrevendo a estrutura interna de seu escalonador padrão (*Credit*). Nesse trabalho o objetivo principal é implementar maneiras de tornar o Xen uma ferramenta de virtualização com um maior desempenho de processamento. Para tanto, analisamos possíveis pontos de otimização do escalonador. Após realizadas as mudanças em seu código-fonte, mostramos avaliações de desempenho deste escalonador mostrando seus resultados.*

1. Introdução

Nos últimos anos, a capacidade de processamento dos computadores tem aumentado consideravelmente. Entretanto, toda a capacidade oferecida não vem sendo utilizada ao máximo. Há situações onde aplicações poderiam ser executadas de uma maneira mais eficiente potencializando o uso de CPU (*Central Processor Unit*). Uma das soluções é o uso da virtualização, que está ganhando cada vez mais destaque, e sendo utilizada com resultados bastante satisfatórios [Mergen et al. 2006].

Este artigo está organizado em 5 seções. Uma introdução sobre virtualização é apresentada na primeira seção. A segunda seção mostra o monitor de máquinas virtuais Xen, com ênfase na estrutura interna de seu escalonador padrão (*Credit*). Na terceira seção veremos as modificações realizadas no código-fonte do escalonador. Em seguida, a quarta seção apresenta avaliações de desempenho do escalonador original versus o escalonador com as modificações propostas. Por fim, mostramos alguns trabalhos futuros na quinta seção e nossas conclusões.

2. Xen

O Xen foi desenvolvido pelo *Systems Research Group* da Universidade de Cambridge [Barham et al. 2003], e é parte de um projeto maior chamado XenoServers, projeto este

que provê um ambiente de computação global distribuída. O Xen permite compartilhar uma simples máquina para vários clientes rodando sistemas operacionais e programas.

O Xen utiliza o conceito de paravirtualização o que possibilita ao sistema operacional que executa sobre uma máquina virtual ter a ilusão de estar sendo executado diretamente sobre o *hardware* [Habib 2006]. O Xen se encarrega de organizar a maioria das requisições feitas pelas máquinas virtuais e repassá-las ao domínio0 como veremos a seguir.

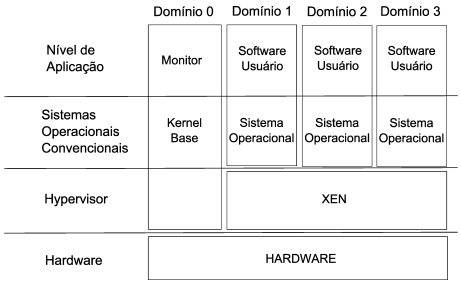

Figura 1. Xen

Na Figura 1, pode-se observar que o Xen se divide em quatro níves. O primeiro nível (nível de aplicação) corresponde às aplicações que rodam dentro das máquinas virtuais (domíniosU) juntamente com os *softwares* adicionais que monitoram e gerenciam o funcionamento do Xen. No segundo nível (sistemas operacionais convencionais) encontram-se o sistema operacional hospedeiro com seu *kernel* modificado para proporcionar virtualização e os sistemas operacionais convidados (máquinas virtuais). No terceiro nível, *hypervisor* (domínio0), está a camada que controla as chamadas de sistema entre as máquinas virtuais e o *hardware*, e por fim, localiza-se o *hardware*.

2.1. *Credit Scheduler*

O SMP Credit [Antoniou and Stavrakakis 2002] é um escalonador construído para manter um trabalho justo para todos os processadores em máquinas SMP. A cada domínio convidado são atribuídos dois valores: um peso e um limite. Um domínio com peso 512 terá duas vezes mais direito à execução que um domínio com peso 256. Os valores de peso variam de 1 a 65535, sendo o padrão 256. O limite geralmente fixa o máximo de CPU (*Central Processor Unit*) que um sistema convidado vai poder consumir, mesmo que o sistema anfitrião tenha ciclos de CPU inativos. O limite de uma CPU é expresso em porcentagem de uma CPU física como: 100 é uma CPU física, 50 é meia CPU, 400 são 4 CPUs, e assim por diante.

O escalonador SMP Credit provê, de forma automática, balanceamento de carga entre as VCPUs (CPUs virtuais) dos domínios convidados, utilizando todas as CPUs de uma máquina SMP. Uma vantagem desse escalonador, é que o administrador não necessita referenciar cada VCPU a cada CPU, o escalonador já trabalha dessa maneira, associando uma VCPU a uma CPU.

Cada CPU controla uma fila local do funcionamento de VCPUs em execução. Esta fila é classificada pela prioridade de cada VCPU. Uma prioridade de VCPUs pode ser um dos dois valores: over ou under, que representam se a VCPU excedeu ou não a sua fatia de acesso justa ao recurso do processador. Enquanto uma VCPU executa, consome

créditos. Assim, frequentemente é recalculada a contabilidade de quantos créditos cada máquina virtual ativa ganhou. Os créditos negativos implicam em uma prioridade *over*.

Até que uma VCPU consuma todos seus créditos, sua prioridade estará *under*. Quando uma CPU não encontra uma VCPU de prioridade *under* em sua fila local de funcionamento, buscará em outra CPU por uma VCPU de prioridade *under*. Assim, garante o balaceamento de carga, onde cada máquina virtual recebe sua parte justa de recursos do processador. Antes que um processador fique inativo, olhará em outro processador para encontrar uma VCPU executável. Isto garante que nenhum processador trabalhe lentamente quando existem execuções a cumprir no sistema. Portanto, o escalonador tem a capacidade de quando uma CPU estiver parada, buscar VCPUs de outras CPUs para execução.

O Credit prima por uma maneira justa de alocar fatias de CPU para os domínios virtuais, porém dependendo do tipo de processos que estiverem rodando nas máquinas virtuais, esse recurso não é distribuído de forma justa. O principal problema está em que o seu tempo de processamento para as máquinas virtuais, que é estático, ou seja 30 milisegundos.

3. Otimização do Código-Fonte

Na Figura 2 podemos observar como funciona o escalonador Credit, e quais etapas são percorridas para realocar suas novas configurações.

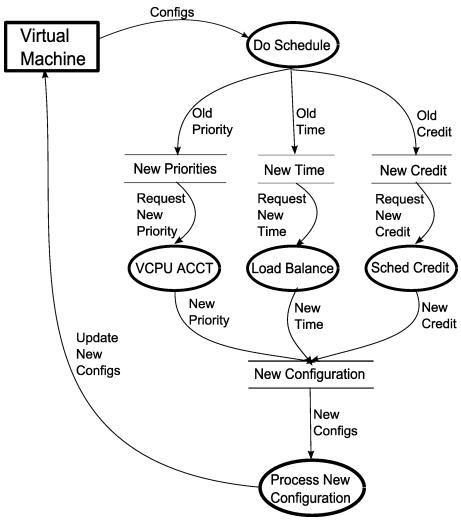

Figura 2. Xen

Primeiramente a máquina virtual (*Virtual Machine*) que será escalonada contém informações como créditos, prioridade do VCPU, e tempo de acesso ao CPU (*configs*), que passa ao *Do Schedule*, realizando o escalonamento. Estas informações são delegadas para o *New Priorities, New Time, New Credit*, que verificam se essas informações estão de acordo com as prioridades, tempo, e créditos do escalonador, e se é necessário mais processamento para a máquina virtual.

Havendo a necessidade de maior fatia de processamento, é feita uma requisição de novos parâmetros (prioridade, tempo e créditos) através dos processos *VCPU ACCT*

(aumenta a prioridade do VCPU), *Load Balance* que determina um tempo de acordo com a prioridade, e o *SCHED CREDIT* que realiza a contagem de créditos, assim realizando uma nova configuração (*New Configuration*).

Com essas novas configurações, é executado um processo de verificação (*Process New Configuration*), que verifica se esta nova configuração irá aumentar o desempenho, aplicando a mesma para a máquina virtual (*update new configs*). Para solucionar o problema de tempo de acesso ao CPU, propomos a redefinição de alguns parâmetros, como o tempo de escalonamento que era estático com 30 milisegundos para cada associação VCPU a domínio. Se o quantum for muito grande, o comportamento será igual a um escalonador *first-in-first-out*, por outro lado, se o quantum for muito pequeno, o desempenho é comprometido pelo número excessivo de preempções e consequentes atrasos para a troca de contexto.

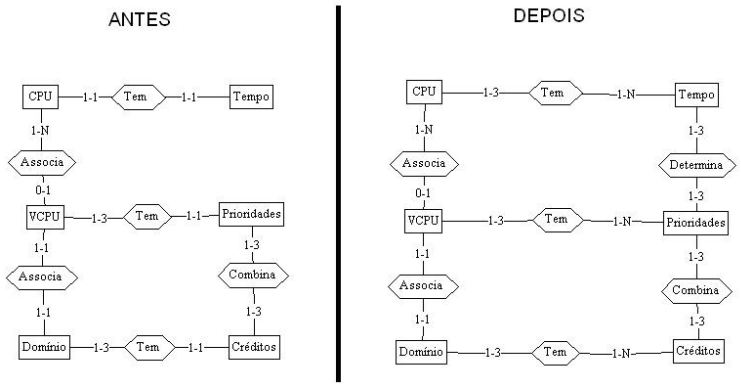

Figura 3. Xen

Na Figura 3, representado no diagrama E-R, vemos a estrutura desse relacionamento antes e depois da solução. Como pode-se observar todo o domínio possui uma VCPU, e todo VCPU possui CPU. Sabemos que cada domínio possui créditos que liga com as prioridades dos VCPU, por exemplo, dependendo do número de créditos, se altera a prioridade do VCPU, mas continuamos sem nenhuma conexão com o tempo de escalonamento, que permanece constante em 30 milisegundos. Ainda na Figura 3, depois da implementação da solução podemos perceber que há uma relação entre prioridades do VCPU com tempo de escalonamento, pois dependendo da prioridade dos domínios, será alterado o tempo de acesso ao CPU.

3.1. *Quantum* Proporcional à Prioridade

Agora são considerados três tipos de tempos, implementados através das seguintes definições:

- O normal, tempo padrão definido pelo Credit (30 milisegundos de acesso ao CPU);
- O máximo, que determina 30% a mais tempo de acesso ao CPU que o definido pelo Credit;
- O mínimo, que lhe concede 30% a menos de tempo de acesso ao CPU que o definido pelo Credit.

Para haver uma ligação entre tempo de escalonamento e a implementação de níveis de serviços, alteramos a contagem de créditos. Por exemplo, se existir uma prioridade

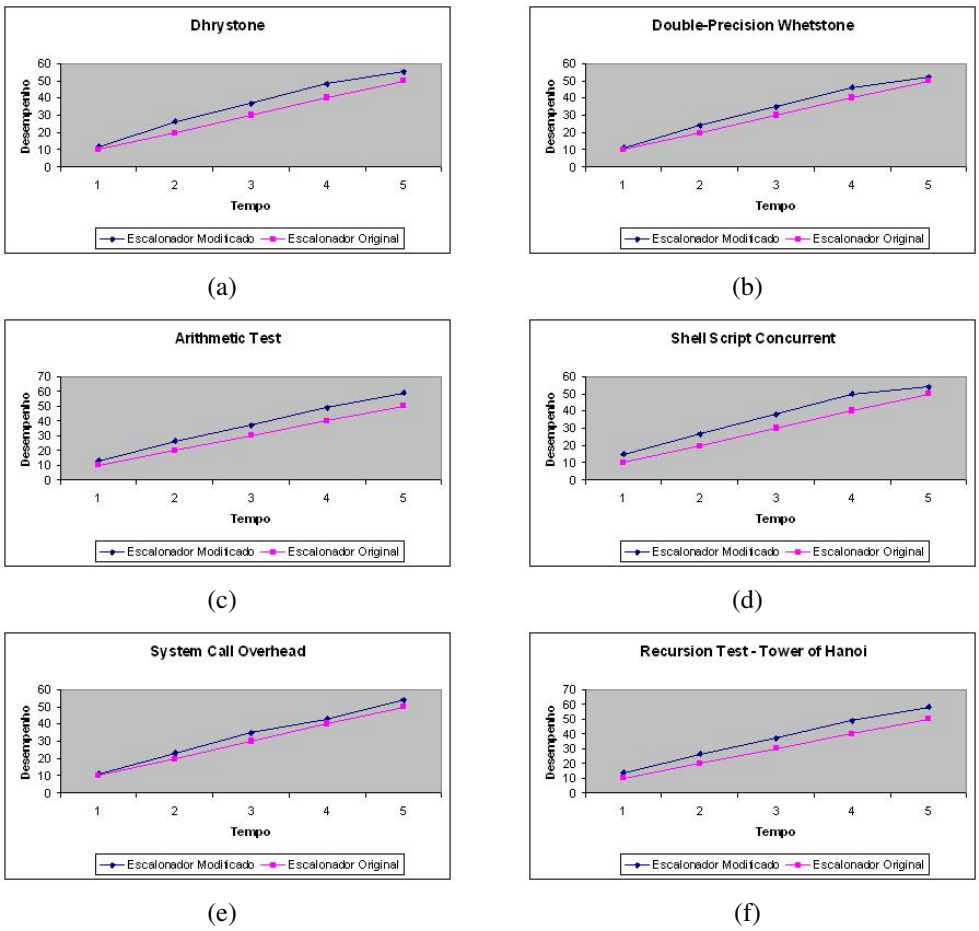

Figura 4. Avaliações CPU-Bound.

máxima de tempo para alguma máquina virtual, então deve haver uma contagem créditos máxima, assim potencializando ainda mais o desempenho das máquinas virtuais. Para processos que se utilizam intensivamente da CPU, essa metodologia se torna interessante, pois notamos em nossas avaliações que um ambiente com processos *CPU-Bound* tem mais desempenho com um quantum maior.

4. Avaliação de Desempenho

Foram realizados testes (execução de *benchmarks*) [Kalibera et al. 2006] sobre as máquinas virtuais, proporcionando formas padronizadas de avaliação do desempenho de sistemas, retirando estatísticas do seu funcionamento. O *benchmark* escolhido foi o Unixbench, utilizado para encontrar gargalos em partes específicas dos subsistemas do *kernel*.

Foram realizadas 200 execuções do Unixbench com o escalonador original e 200 execuções com o escalonador modificado. Para manter um intervalo de confiança, os 10% maiores e os 10% menores resultados foram excluídos da média final. Os resultados apresentam os valores alcançados em cada métrica *versus* o tempo decorrido na execução em minutos.

Na Figura 4 vemos o resultado de testes que utilizam intensivamente a CPU, tais como *Dhrystone* (a) que consiste no número de iterações de um laço por segundo, *Whets-*

tone (b) que consiste em operações de ponto flutuante, *Arithmetic Test* (c) que utiliza adição, multiplicação, subtração e divisão, *Shell Script Concurrent* (d) que testa a concorrência entre processos , *System Call Overhead* (e) que testa a latencia das chamadas de sistema e *Recursion Test - Tower of Hanoi* (f) que serve para testar soluções que necessitam de recursividade.

5. Trabalhos Futuros

Como trabalhos futuros, temos agora várias possibilidades, dentre elas, permitir o ajuste dos critérios de escalonamento em função do comportamento dos processos. Para tanto, deve ser criado um histórico desse comportamento e conforme esse histórico, ajustar o melhor quantum.

6. Conclusões

O escalonador Credit, como padrão possui apenas uma definição de tempo (30 milisegundos) para cada domínio, tornando as máquinas virtuais, de uma forma geral, igual uma perante a outra em questão de acesso ao CPU. Conforme a Análise do escalonador observou-se isso, e então foi proposta uma solução. Essa solução trabalha a idéia de tornar as máquinas virtuais única para CPU.

Para tornar as máquinas virtuais diferentes em relação a tempo de acesso ao CPU, a implementação propõe novas prioridades ao sistema, adicionando definições máxima e mínima na contagem de créditos e de tempo. As definições implementadas criam tempos dinâmicos de acesso ao CPU, pois sempre que alguma prioridade for alterada, ou existir necessidade de maior desempenho, seu tempo de acesso modificará. Conforme a Análise dos resultados, utilizando o *benchmark* Unixbench, observamos que houve uma melhora significativa de 12% a mais de desempenho com as modificações propostas para processos *CPU-Bound*.

7. Agradecimentos

Avelino Zorzo - FACIN/PUCRS e HP Brasil.

Referências

Antoniou, Z. and Stavrakakis, I. (2002). An efficient deadline-credit-based transport scheme for prerecorded semisoft continuous media applications. *IEEE/ACM Transactions on Networking*, 10(5):630–643.

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. (2003). Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, Bolton Landing, NY, USA. ACM Press.

Habib, I. (2006). Xen. *Linux Journal*, 2006(145):4.

Kalibera, T., Lehotsky, J., Majda, D., Repcek, B., Tomcanyi, M., Tomecek, A., Tuma, P., and Urban, J. (2006). Automated benchmarking and analysis tool. In *VALUETOOLS '06: Proceedings of the 1st international conference on Performance evaluation methodolgies and tools*, page 5, Pisa, Italy. ACM.

Mergen, M. F., Uhlig, V., Krieger, O., and Xenidis, J. (2006). Virtualization for high-performance computing. *SIGOPS Operating System Review*, 40(2):8–11.