Comparação de Estratégias de Paralelização em Ambientes Paravirtualizados

Artur Baruchi, Edson Toshimi Midorikawa
Laboratório de Arquitetura e Computação de Alto Desempenho – Escola Politécnica – Universidade de São Paulo
{artur.baruchi,edson.midorikawa}@poli.usp.br

Resumo

Diversas pesquisas vêm sendo realizadas na tentativa de minimizar o overhead da Virtualização em ambientes de alto desempenho. Os benefícios de se utilizar virtualização, tais como a melhor utilização de recursos computacionais e a maior facilidade em se criar e destruir Máquinas Virtuais de acordo com as necessidades do ambiente, tornam esta tecnologia bastante atraente em ambientes de alto desempenho. Neste artigo são realizados testes com MPI e OpenMP, que são duas tecnologias empregadas em ambientes de alto desempenho, na tentativa de identificar qual delas sofre maior degradação de desempenho utilizando-se Paravirtualização. Os testes foram realizados, com três benchmarks de características diferentes, sendo que um deles utiliza computação intensiva (Multiplicação de Matriz) e o outro maior número de troca de mensagens (Cálculo de dissipação de Calor). O terceiro benchmark utilizado foi o Integer Sort (IS), que faz parte do NPB (NAS Parallel Benchmark). Como Monitor de Máquinas Virtuais foi utilizado o Xen, pois disponibiliza maior flexibilidade na configuração.

Os resultados demonstraram que a utilização do MPI em ambientes paravirtualizados sofreu pouca degradação de desempenho e em determinadas situações apresentou melhora no desempenho em comparação com um ambiente utilizando Linux Nativo. Em contrapartida o OpenMP apresentou maior sensibilidade ao utilizar-se Máquinas Virtuais.

1. Introdução

Diversas propostas para melhorar o desempenho de Máquinas Virtuais (MV) podem ser encontradas na literatura [1] [2] [3] [4] [5]. É notório que na maioria das pesquisas encontradas com foco no desempenho em MV é abordado de alguma forma operações de I/O. Isto se deve principalmente à camada de software adicional[1], entre a MV e o Hardware, inerente à virtualização. Esta camada acaba trazendo um custo maior em operações de I/O do que em operações relativas a processamento.

A forma como é realizada a virtualização influencia diretamente no desempenho das MV's. As técnicas de virtualização que possuem maior desempenho em ambientes de alto desempenho são a Paravirtualização e a Virtualização na Camada do Sistema Operacional [6].

A Paravirtualização consiste em modificar o Sistema Operacional a ser instalado na MV (ou Sistema Operacional Convidado) de tal forma que este utilize estruturas disponibilizadas pelo MMV em operações de I/O. Já a Virtualização na Camada do Sistema Operacional, não necessita de um MMV. O Sistema Operacional real é modificado para isolar de forma segura múltiplas instancias de um Sistema Operacional. Neste tipo de virtualização não há necessidade de realizar *trap's* para o MMV, um dos maiores causadores do *overhead* na virtualização [7].

Neste trabalho foi utilizada a técnica de Paravirtualização, representada pelo Xen [8]. Esta estratégia de virtualização foi escolhida por causa de sua maior abstração e melhor isolamento das Máquinas Virtuais. Dentre as implementações de Paravirtualização, o Xen foi utilizado, pois o acesso ao seu código fonte é livre e possui maior flexibilidade em sua configuração.

Os testes realizados neste artigo têm o objetivo de comparar e identificar, qual das duas técnicas de paralelização (MPI ou OpenMP) possui maior ou menor sensibilidade em um ambiente virtualizado. Durante os testes variamos algumas configurações das Máquinas Virtuais (como programa de escalonamento e quantidade de CPU's Virtuais) com o objetivo de

[1] Esta camada de Software adicional, também é conhecida como Monitor de Máquinas Virtuais (MMV) ou *Hypervisor*.

observar a influência destas configurações no desempenho dos *benchmarks*.

O presente artigo está dividido da seguinte maneira: Na seção 2, são abordados alguns trabalhos relacionados e a seção 3 descreve as estratégias de paralelização que serão usadas no decorrer deste trabalho. Posteriormente, na seção 4, são demonstrados os resultados obtidos, em seguida, na seção 5, é realizada a análise dos resultados. Na seção 6, o artigo é finalizado com as conclusões e com a descrição da contribuição deste trabalho.

2. Trabalhos Relacionados

A quantidade de pesquisas realizadas utilizando-se Máquinas Virtuais e sistemas de alto desempenho mostra que as características da virtualização são atrativas para um ambiente de alto desempenho. Dentre as características que mais chamam a atenção estão a facilidade de criação e recuperação de uma Máquina Virtual e a melhor utilização de recursos computacionais. Estas características fazem da virtualização uma das grandes esperanças em aumentar a disponibilidade de ambientes críticos sem a necessidade de grandes investimentos. A utilização de forma mais eficiente os recursos já existentes justificam a quantidade de pesquisas e trabalhos realizados com virtualização e alto desempenho.

O trabalho realizado por Neves et. al. [9], mostra como a flexibilidade na criação e recuperação de MV's pode ser de grande utilidade em um ambiente distribuído. Foi demonstrado que a migração de uma MV para outro nó físico de um cluster é viável quando a aplicação é executada por um longo período. Em aplicações mais rápidas, a migração causa muitos impactos no tempo de execução.

Outro trabalho bastante interessante realizado por Youseff et. al. [10], identifica o impacto ao se utilizar a Paravirtualização em execuções com MPI. Neste trabalho os autores dividem as execuções MPI em duas partes, a parte de comunicação e a parte de computação. Foi identificado que o maior gargalo em aplicações MPI é a parte de comunicação quando o tamanho das mensagens é menor. Nesta situação, o *overhead* causado pela estrutura de comunicação entre a MV e o MMV é mais perceptível.

No artigo desenvolvido por Walters et. al. [6], os autores fazem um estudo sistemático de desempenho em vários pontos como Sistema de Arquivos e Rede. Porém o foco principal deste trabalho é a escalabilidade dos programas MPI em diversas implementações de MV's (como VMWare, Xen e OpenVZ) SMP. O artigo conclui que a Paravirtualização (representada pelo Xen) e a Virtualização na camada do Sistema Operacional (representada pelo OpenVZ) são as duas técnicas de virtualização que possuem menor *overhead* ao utilizar MPI.

Por fim, no trabalho confeccionado por Pan et. al. [11], os autores realizam testes usando MV's geograficamente distribuídas na Internet utilizando uma aplicação *Peer-to-Peer*. Neste mesmo trabalho, os autores desenvolvem um *socket* que utiliza uma técnica de virtualização na camada de usuário (UML, *User Mode* Linux). Com este novo *socket*, os autores conseguem aumentar de forma substancial a vazão em um ambiente virtualizado.

Diferente dos trabalhos apresentados acima, o presente artigo tem a finalidade de comparar duas tecnologias utilizadas nos sistemas de alto desempenho em MV's e avaliar em quais situações a virtualização pode ser utilizada de tal forma que melhore a utilização dos recursos computacionais disponíveis, mas sem causar danos ao desempenho nas aplicações.

3. Análise de Estratégias de Paralelização

Um dos principais objetivos deste trabalho é avaliar qual é a melhor estratégia de paralelização em um ambiente paravirtualizado. A primeira forma de paralelização faz uso de memória distribuída, este é representado pelo MPI (*Message Passing Interface*). A segunda forma de paralelização usada foi com a utilização de memória compartilhada, representada pelo OpenMP.

Para análise das estratégias foram utilizados três *benchmarks*. O primeiro programa de Multiplicação de Matrizes [12] foi usado pela sua característica de computação intensiva. Este programa divide a matriz em partes iguais para os processos escravos e espera o resultado de cada um deles para contabilizar o resultado final. O segundo programa, utilizado para cálculo de dissipação de calor (conhecido como *Heat*[13]), tem característica de realizar intensas trocas de mensagens, visto que o cálculo de um determinado setor depende do cálculo dos setores vizinhos. O terceiro benchmark foi o IS (*Integer Sort*) classe C, disponibilizado como parte do conjunto de benchmarks do NPB 3.3 [16] (*Nas Parallel Benchmark*).

Para melhor entendimento do comportamento dos testes realizados, foi criado um modelo para cada um dos programas usados como *benchmark*. Os modelos e as estratégias de paralelização utilizadas neste artigo serão apresentadas nas subseções a seguir.

3.1. Tipos de Paralelização

Nesta subseção são descritas de forma breve a biblioteca MPI e a API OpenMP que são usadas frequentemente na paralelização de aplicações de alto desempenho.

O MPI (*Message Passing Interface*) é um protocolo de comunicação, independente de linguagem de programação, usada para computação paralela. Aplicações MPI normalmente tem a estrutura mestre-escravo e as tarefas paralelas são distribuídas pelos nós de processamento. A comunicação entre elas usa primitivas de troca de mensagens do tipo *send* e *receive*.

As implementações do MPI consistem em um conjunto de subrotinas, que são chamadas a partir da linguagem de programação em uso. As maiores características do MPI são a portabilidade e também a velocidade. Existem duas implementações do MPI, a versão 1.2 (conhecida como MPI-1), que tem foco principalmente na passagem de mensagens e tem um ambiente de execução estático. A versão 2.1 (MPI-2), possui novas características, como I/O paralelo, gerenciamento de processo dinâmico e operações de memória remotas.

O OpenMP (*Open Multi-Processing*) é uma API (*Application Programming Interface*) que suporta multiprocessamento com a utilização de memória compartilhada. Basicamente, o OpenMP inclui um conjunto de diretivas do compilador e variáveis de ambiente que definem o comportamento do programa em tempo de execução.

Estas diretivas são definidas por um grupo de fornecedores de Hardware e Software e foi criada com a intenção de ser um modelo portável e escalável. Pode-se dizer que o OpenMP é uma implementação de *multithreading*. O ambiente de execução aloca cada *thread* a um processador dependendo da utilização, da carga da máquina e de outros fatores. A quantidade de *threads* a ser executada pode ser definida em tempo de execução, utilizando variáveis de ambiente ou pode ser definida diretamente no código fonte do programa.

3.2. Modelagem dos Programas de *Benchmark*

Os programas usados como *benchmark*, multiplicação de Matriz e *Heat*, utilizam matrizes de tamanho 500x500, tanto na versão MPI como na versão OpenMP. O *Heat* possui outro parâmetro que influencia no tempo de execução, que é a quantidade de passos que o programa executará para realizar o cálculo de dissipação. Quanto maior o número de passos, maior é a precisão e maior é o tempo necessário para cálculo, tanto na versão MPI como na versão OpenMP foi utilizado 500 passos para o cálculo. Com esta quantidade de passos, o tempo de execução se mostrou razoável para as duas versões.

No programa de Multiplicação de Matrizes, dada uma matriz A e uma matriz B, o processo mestre divide estas duas matrizes pelo número de escravos e as distribui de forma homogênea. Os processos escravos recebem estes dados e realizam o processamento necessário e após este cálculo devolvem o resultado para o processo mestre que consolida os resultados.

O segundo programa utilizado realiza o cálculo de dissipação de calor em uma superfície metálica. Neste programa, o processo mestre divide os dados (no caso uma matriz que representando a temperatura de cada uma das regiões de um objeto qualquer). Os processos escravos recebem os dados e realizam um determinado cálculo, após isto, os processos escravos trocam informações com seus vizinhos imediatos. Esta troca de informações se repetirá pela quantidade de passos determinados no programa.

O terceiro *benchmark* usado neste artigo foi o IS que é parte do NPB. Este *benchmark* é amplamente utilizado no meio acadêmico para avaliar o desempenho de sistemas computacionais de alto desempenho. Cada um dos *benchmarks* que compõe o NPB tem uma classe, que define a grandeza dos programas e aumenta a carga de trabalho, no presente trabalho foi usada a classe C do programa IS.

O programa IS é baseado no algoritmo de ordenação *Bucket Sort* [27], este programa envolve comunicação intensiva de todos-para-todos (*all-to-all*) misturando mensagens de tamanho pequeno e grande. Outra característica interessante do IS é a sua sensibilidade em relação a largura de banda e também a latência.

3.2. Execução dos Testes

Cada um dos *benchmarks* foi executado variando algumas configurações, como quantidade de Processadores Virtuais e a quantidade de *threads* (no caso do OpenMP) ou quantidade de processos (no caso do MPI). Foram realizadas outras alterações no ambiente virtualizado como forma de analisar os resultados obtidos nos testes primários.

A avaliação de desempenho neste trabalho foi mensurada a partir do tempo de execução de cada um dos benchmarks. O tempo foi obtido a partir do comando *time*[17] do Linux. A partir da saída deste comando é possível verificar o tempo real que um determinado programa executou e também o tempo

gasto executando chamadas de sistema e executando o próprio código.

O primeiro teste foi realizado com uma máquina real executando Linux Nativo (LN) para marcar um ponto de referência para os testes posteriores. Os testes utilizando MPI foram executados com dois nós e depois executados localmente, isto é, sem a utilização de outros nós.

O segundo teste foi executado em uma Máquina Virtual (MV) com dois processadores Virtuais e 1 Gb de memória. O objetivo deste teste é comparar o desempenho da MV com a Máquina Real em condições parecidas. Posteriormente, outros processadores virtuais foram adicionados para verificar a escalabilidade.

Por fim, no terceiro teste foram criadas duas MV's com um processador virtual cada e 1 Gb de memória. Desta forma, cada uma das MV's utilizou um processador real. O principal objetivo deste teste é verificar se existe algum ganho de desempenho ao utilizar as estruturas de comunicação disponibilizadas pelo Xen[8] ao executar programas paralelos. Neste teste apenas os programas MPI foram executados.

4. Análise de Desempenho

Os testes foram realizados com duas Máquinas interligadas por um cabo *cross* a 100 Mb/s *Full duplex*. Utilizando cabo *cross*, garantimos que não há influência de outros equipamentos de rede, como roteadores ou *switchs*. A máquina principal, onde foram executados os testes de comparação de um ambiente *stand alone*, possui processador *dual core* AMD Athlon 64 Bits de 2 GHz e 6 GB de memória. A máquina secundária possui um processador *dual core* AMD Athlon 64 Bits de 2.2 GHz e 1 Gb de memória.

Ambas as máquinas estavam com o Fedora Core 8 instalado, usando o Kernel 2.6.24. O MMV utilizado foi o Xen [15] versão 3.1.2. Nos testes, foram utilizados o OpenMP, implementado no gcc, cuja versão utilizada foi a 4.1.2. A versão do MPI utilizada nos testes foi a LAM/MPI [14] 7.1.2.

4.1. Programa de Multiplicação e *Heat*

A primeira comparação realizada foi entre a Máquina Real e uma Máquina Virtual com dois processadores virtuais usando MPI. Este mostrou que programas MPI em MV's teve um desempenho 17% melhor no programa de multiplicação de matriz e 14% melhor no programa *Heat*.

Analisando as versões OpenMP, verificou-se que o programa de Multiplicação de Matrizes também apresentou desempenho 10% melhor que no LN. Os resultados foram diferentes em relação ao programa *Heat*. No ambiente virtualizado o seu desempenho foi 8% pior. Na seção seguinte serão analisadas as causas destes resultados.

No teste com o OpenMP, aumentamos o número de processadores virtuais na MV para 4 e 8. A degradação de desempenho foi bastante acentuada em relação ao primeiro teste, quando foram utilizados 2 processadores virtuais. Utilizando 4 processadores, o desempenho foi 88% pior no *Heat* e 86% pior ao utilizar o programa de multiplicação. Com a MV configurada com 8 processadores, o tempo de execução do programa *Heat* foi 355% pior e a multiplicação 321% pior.

Aumentando o número de processadores virtuais nos testes com MPI, mas sem aumentar o número de processos escravos, o tempo de execução se manteve estável. Ao aumentar o número de escravos, porém mantendo dois processadores virtuais o desempenho utilizando MV foi superior (em média 8% superior para o programa de multiplicação e 33% para o *Heat*). Na tabela 1 é apresentada a comparação de tempo de execução.

Tabela 1. Comparação de desempenho (em segundos) entre Máquina Virtual e o Linux Nativo executando a versão MPI dos Programas.

Numero de Escravos	MM-mv	MM-LN	Heat2D-mv	Heat2d-LN
1	2.70	2.75	7.57	7.5
2	2.70	2.93	4.53	8.33
3	2.23	2.20	5.53	6.07
4	2.20	2.67	4.67	7.43
5	2.43	2.33	5.20	6.43
6	2.13	2.53	4.63	7.67
7	2.33	2.20	5.03	6.87
8	2.37	2.83	4.70	8.10
9	2.47	2.33	5.00	7.30
11	2.30	2.53	5.00	7.87
23	2.50	3.20	5.20	11.23
35	3.00	4.13	5.50	14.60

No próximo teste, foram criadas duas MV's na mesma máquina física, cada uma utilizando um processador virtual. Os resultados deste experimento também foram favoráveis a MV. Nota-se que houve melhora substancial no programa de multiplicação nesta configuração, pois o desempenho para este programa foi em média 20% melhor. O segundo programa continuou com melhor desempenho, porém teve ligeira queda no tempo de execução, em média o desempenho foi 21% melhor no ambiente virtualizado.

4.2. Benchmark *Integer Sort* (NPB)

Assim como os primeiros experimentos utilizando os programas de multiplicação e de dissipação de calor, o primeiro teste foi realizado utilizando-se uma máquina real com Linux nativo executando MPI localmente, em apenas um nó. A comparação direta entre a Máquina Real e uma MV com o mesmo número de processadores virtuais e mesma quantidade de escravos (no caso 2 escravos, um para cada processador) o desempenho da Máquina Real foi 33% melhor.

No próximo experimento, o número de processos MPI foi alterado, variando de 2 a 32, porém ainda executando em apenas um nó, tanto na MV quanto na Máquina Real, os resultados se mantiveram. Em média o desempenho da Máquina Real foi 32% melhor que a MV.

Tabela 2. Comparação de desempenho (em segundos) entre Máquina Virtual e o Linux Nativo executando a versão MPI do *benchmark* IS (NPB).

Numero de Escravos	Tempo (Seg)-1 MV	Tempo (Seg)-2 MV	Tempo (Seg)-Linux Nativo
1	54.93	54.33	41.17
3	57.2	56.90	42.53
7	60.67	60.00	46.33
15	66.33	66.33	53.50
31	93.33	95.67	67.07

Na tabela 2 é interessante notar que a diferença de desempenho entre uma MV com dois processadores Virtuais e duas MV's com um processador Virtual cada foi praticamente desprezível.

A versão OpenMP do *benchmark* IS apresentou comportamento diferente da versão MPI. No geral a Máquina Real com Linux Nativo teve pior desempenho. Com a mesma quantidade de processadores virtuais, o desempenho da MV foi em média 3% melhor. O melhor desempenho identificado foi utilizando quatro threads, nesta situação o desempenho da MV superou em 9% a Máquina Real.

Variando o número de processadores virtuais para 4 e 8, houve degradação no desempenho da MV. Com 4 processadores virtuais, o desempenho foi ligeiramente superior ao da Máquina Real. A melhor situação identificada com maior número de processadores virtuais, foi utilizando duas *threads* e 4 processadores virtuais. Na figura 1, é apresentado um gráfico com os desempenhos relativos da versão OpenMP do IS.

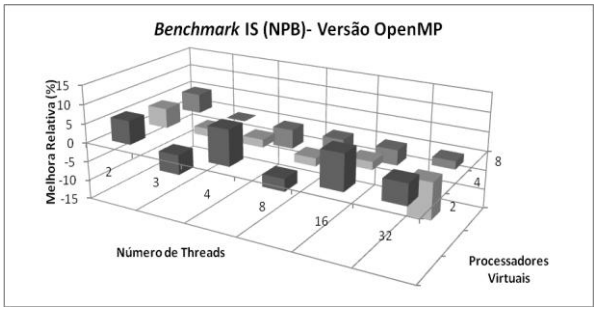

Figura 1. Desempenho do *Benchmark* IS na versão OpenMP.

5. Análise Dos Resultados

Nas primeiras comparações, utilizando os programas de multiplicação de matriz e de dissipação de calor, o desempenho utilizando MV's na versão MPI teve melhor desempenho tanto utilizando duas MV's com um processador virtual cada. Para analisar o que ocorre nestas situações isolamos os programas e as MV's em duas partes: A parte de Comunicação e a parte de Computação.

5.1. Comunicação

Nesta versão do Xen, as MV's (que não possuem acesso privilegiado) se comunicam com o IDD (*Isolated Driver Domain*) através de Anéis Descritores (ou *Rings Descriptors*) [18]. Basicamente a comunicação entre MV's ocorre utilizando três mecanismos [19]: *event-channel*, *grant tables* e o *Xenstore*.

No caso do primeiro experimento, a comunicação entre as MV's é realizada através da rede, nesta situação o caminho realizado entre uma MV para a outra é ineficiente, pois utiliza a pilha TCP/IP além de realizar diversas chamadas a *Hypercalls*[2] para invocar o mecanismo de *page flipping* do Xen. Para tentar melhorar a comunicação entre as MV's, o Xen suporta uma operação atômica (*atomic operation*) [8] que implementa uma transmissão de rede sem realizar cópia de dados entre as MV's. Esta técnica é bastante útil em aplicações que necessitam trocar mensagens de tamanho pequeno, porém mesmo assim o desempenho

[2] *Hypercall*, é uma chamada de sistema feita por uma MV ao MMV.

da comunicação entre as MV's é bastante inferior se comparada com um ambiente real [20].

Ao analisar a parte de comunicação entre as MV, não é difícil observar que devido à complexidade das suas estruturas de comunicação o melhor desempenho das MV's não é devido à comunicação. Para verificar o desempenho das primitivas básicas MPI, foi utilizado o benchmark da Intel IMB, versão 3.1 (*Intel MPI Benchmark Suite*) [21].

O IMB disponibiliza diversos testes, porem foi utilizado somente os testes de *PingPing*, *PingPong* e *SendReceive*, pois os programas dos testes fazem uso somente do MPI_Send e MPI_Receive. Na figura 2 são apresentados os resultados do IMB.

Com estes resultados, descartamos que as complexas estruturas de comunicação entre MV's sejam responsáveis pelo melhor desempenho nos primeiros experimentos.

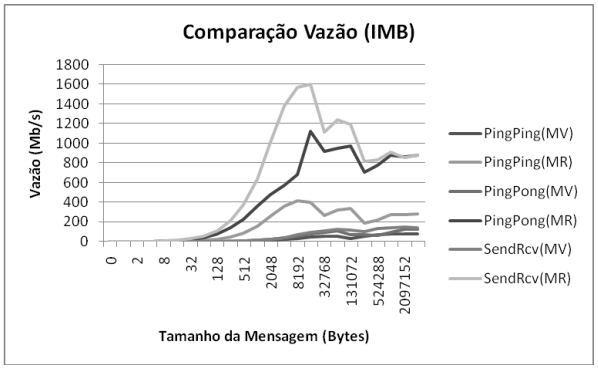

Figura 2. Comparação da Vazão ao usar o benchmark IMB (MV: Máquina Virtual e MR: Máquina Real)

No caso da versão OpenMP, verificamos dois resultados diferentes, no caso do programa *Heat*, o desempenho da MV foi pior, porém no programa de Multiplicação de Matrizes a MV apresentou melhor desempenho. Diferente do MPI, o OpenMP utiliza memória compartilhada para realizar a comunicação entre as *Threads*.

No Xen, o gerenciamento de memória é feito pelo próprio Sistema Operacional instalado na MV. Algumas operações realizadas pela MV, como a atualização da tabela de páginas, devem ser validadas antes pelo MMV, por esta razão o excesso de paginação pode causar maior *overhead* em um ambiente virtualizado. Além disse, quando uma nova página é necessária, a MV deve reservar e registrá-la no Xen, a partir deste ponto todas as atualizações nesta página devem ser validadas pelo Xen.

Ao observar os dois programas, nota-se que o programa *Heat*, realiza maior número de troca de mensagens e de atualizações. O outro programa, de multiplicação, realiza menor número de troca de mensagens. Devido a maior quantidade de troca de mensagens do programa *Heat*, teve seu desempenho prejudicado. Nas duas tecnologias, a comunicação entre MV's não influencia de forma positiva o desempenho. O próximo passo é avaliar a computação nos dois programas.

5.2. Computação

Os binários utilizados tanto no Linux Nativo como nas MV's são os mesmos e não foi utilizado nenhum mecanismo de otimização nas MV's. Deste modo, a única diferença na utilização do processador nos dois ambientes é forma como os processos são agendados (escalonados).

O escalonador padrão nesta versão do Xen é o *Credit Scheduler* [22]. Este escalonador possui um mecanismo de balanceamento de carga entre os processadores, desta maneira ele consegue utilizar melhor os recursos da máquina. Cada processador físico possui uma fila de processadores virtuais e cada um destes processadores possui uma determinada prioridade. Caso o processador não possua um processador virtual para processar ele busca na fila dos outros processadores físicos por algum processador virtual pronto para execução.

Na atual versão do Xen possui outro programa de escalonamento, o SEDF [23] (*Simple Earliest Deadline First*), porem este não possui nenhuma otimização para um ambiente SMP. Como comparação foi executado alguns testes utilizando o SEDF e o desempenho foi igual ao da Máquina Real.

Em um teste adicional, usando o benchmark IS, foi executado o MPI com dois e quatro processos. Com dois processos e utilizando o *SEDF*, executou apenas 65% do total de operações por segundo (Mops) que quando executado com o *Credit*. Ao ajustar para quatro processos, a quantidade de operações por segundo com o *SEDF* foi de 46% quando comparado com *Credit*.

Esta analise mostra que a forma como o Xen escalona os processadores virtuais é um diferencial no desempenho quando se trata de um ambiente SMP. Em aplicações *CPU Bound* a utilização de MV's pode ser bastante atrativa.

5.3. Desempenho com o Dom0

Durante as análises deste trabalho, foram realizados alguns experimentos com o Dom0 para verificar sua

influência no desempenho das MV's. O Dom0 (Domínio 0), é uma MV criada pelo Xen no boot da máquina real, este domínio possui acesso ao *device drivers* reais e também o privilegio de criar e destruir as MV's.

Os resultados dos testes no Dom0 foram surpreendentes, em todos os testes ele apresentou melhor desempenho que a Máquina Real e as MV's. No benchmark IS, usando MPI a MV apresentou pior desempenho que a Máquina Real, porém no experimento utilizando o Dom0 o desempenho chegou a melhorar 20% se comparado com a Máquina Real.

Usando outro benchmark, o IMB da Intel, observamos que o Dom0 possui um desempenho superior com tamanhos de mensagens maiores que 16Kb, com mensagens de tamanho menor a vazão do Dom0 foi menor. A princípio credita-se este melhor desempenho a duas coisas: o escalonador *credit scheduler* e aos *descriptors rings* [8].

A utilização dos processadores do Dom0 também é regida pelo *Credit Schedulers* e como foi visto no nesta seção este escalonador possui excelente desempenho em sistemas SMP com o uso de balanceamento de carga. Na Máquina Real, o MTU em todos os testes estava ajustado para 1500 *Bytes* e pacotes acima deste tamanho gera maior número de interrupções e por esta razão pioram o desempenho. Com a utilização dos *descriptors rings* esta limitação não se aplica, pois as transmissões de dados utilizando as estruturas destes anéis podem mover quantidade de dados maiores para o Dom0 causando menor quantidade de interrupções.
Obs.: acrescentar gráfico sedf x credit.

6. Conclusão

As MV's apresentaram, em geral, desempenho satisfatório nas tecnologias de alto desempenho testadas. Em algumas situações o desempenho foi até superior ao das Máquinas Reais.

Para análise, dividimos os programas em duas partes, a comunicação e a computação. As estruturas de comunicação das MV's são bastante complexas e onerosas ao desempenho de aplicações que necessitam realizar troca de mensagens. Porém, na parte de computação foi identificado que o escalonador utilizado pelo Xen tem um impacto bastante positivo no desempenho de sistemas SMP por conta do balanceamento de carga. Na figura 3, pode-se observar a diferença de desempenho entre os dois escalonadores disponível no Xen.

A utilização de programas, tanto MPI como OpenMP em um ambiente virtual pode ser bastante atrativa quando o programa realizam menor número de troca de mensagens, como verificado ao analisar o programa de multiplicação de matrizes. Programas com número maior de troca de mensagens tendem a ser mais prejudicados por sua maior dependência das complexas estruturas de comunicação inerentes à virtualização.

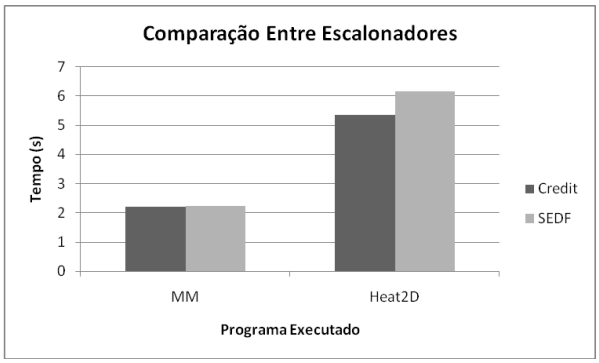

Figura 3. Comparação de Desempenho entre o SEDF e *Credit Scheduler*.

Programas que utilizam OpenMP com muitas trocas de mensagens apresentaram maior sensibilidade em ambientes virtualizados, principalmente devido às estruturas de gerenciamento de memória. Os programas MPI também apresentam a mesma sensibilidade, porém devido às estruturas de comunicação das MV's. Contudo os programas MPI apresentaram melhor comportamento na paravirtualização com o aumento de processadores virtuais ou quantidade de processos. Os programas OpenMP tiveram maior degradação no desempenho com o aumento de processadores virtuais e também com o programa *Heat*, que utiliza maior quantidade de troca de mensagem.

O aumento de processadores virtuais não melhora o desempenho dos programas no Xen. Isto é devido aos programas de escalonamento que trabalham com o escalonamento de processadores virtuais e não com a MV inteira (por exemplo, o *time slice* é atribuído a cada um dos processadores virtuais e não a todos os processadores de uma MV), desta forma o aumento de processadores virtuais causa congestionamento nas filas dos processadores reais.

Como trabalho futuro, pretendemos analisar a influência dos diversos escalonadores e seus parâmetros para melhorar o desempenho das MV's em um ambiente de alto desempenho. Como identificado em Baruchi et. al. [26] a influência dos parâmetros dos escalonadores influênciam de forma significativa o desempenho das MV's. Outro trabalho a ser desenvolvido, seria a implementação de um escalonador de tempo real, desta forma uma MV poderia ter maior prioridade sobre as outras sempre

que precisasse executar algum programa de maior criticidade.

A contribuição deste artigo foi identificar que em determinadas situações a paravirtualização pode trazer ganhos significativos a um ambiente de alto desempenho. Um dos maiores empecilhos em usar a virtualização em ambientes de alto desempenho é justamente a degradação do desempenho causado pela camada de software adicional entre o Sistema Operacional convidado e o *Hardware*, principalmente em operações de I/O.

7. Referências

[1] Willmann, P. Shafer, J. Carr, D. Menon, A. Rixner, S. Cox, A.L. Zwaenepoel, W., "Concurrent Direct Network Access for Virtual Machines", *High Performance Computer Architecture*, IEEE, Scottsdale, AZ, Feb. 2007, pp. 306-317.

[2] D. Gupta, R. Gardner, and A. Vahdat. "Enforcing Performance Isolation Across Virtual Machines in Xen", *Proceedings of the Seventh International Midleware Conference, Melborne, Australia*, Dec. 2006.

[3] Chadha, Vineet I., Ramesh, I., Ravi, M., Jaideep, N., Newell, J. F., Renato, " I/O processing in a Virtualization platform: a simulation-driven approach", *Proceedings of the 3rd International conference on Virtual execution ennvironments*, ACM New York, California, USA, 2007, pp. 116-125.

[4] Menon, Aravind A. L., Cox, "Optimizing Network Virtualization in Xen", *Usenix Annual Techinical Conference*, USA, 2006, pp. 15-28.

[5] L., Jiuxing H., Wei A., Bulent K. P., Dhabaleswar, "High Performance VMM-bypass I/O in Virtual Machines", *Preceedings of the annual conference on USENIX '06*, Usenix Association, 2006, pp. 3-3.

[6] Chaudhary, V. Minsuk Cha Walters, J.P. Guercio, S. Gallo, S., "A Comparison of Virtualization Technologies for HPC", *Advanced Information Networking and Applications*, IEEE, Okinawa, March 2008, pp. 861-868.

[7] Nakajima, Jun K. Mallick, Asit, "Hybrid-Virtualization: Enhanced Virtualization for Linux", *Linux Symposium*, Ottawa, Ontario, Jun. 2007, pp. 87-96.

[8] Barham, Paul Dragovic, Boris Fraser, Keir Hand, Steven Harris, Tim Ho, Alex Neugebauer, Rolf Pratt, Ian Warfield, Andrew, "Xen and the Art of Virtualization", *ACM Symposium on Operating Systems Principles*, ACM, NY,USA, 2003, pp. 164-177.

[9] Veiga N., Marcelo Rosa R., Rodrigo Maillard, Nicolas O. A. Navaux, Philippe, "Impacto da Migração de Máquinas Virtuais de Xen na Execução de Programas MPI", *WSCAD*, Rio Grande do Sul, Gramado, 2007.

[10] Youseff, Lamia Wolski, Rich Gorda, Brent Krintz, Chandra, "Evaluating the Performance Impacto of Xen on MPI and Process Execution for HPC System", *Virtualization Technology in Distributed Computing*, ACM, Washington, USA, 2006.

[11] Pan, Zhelong Ren, Xiaojuan R., Eigenmann Xu, Dongyan, "Executing MPI programs on Virtual machines in na Internet Sharing System", *Parallel and Distributed Processing Symposium*, ACM, April 2006.

[12] Código fonte da Multiplicação de Matrizes: HTTP://www.abaruchi.com/programacao/multiplicacao Matriz .html

[13] Código fonte do Heat2D: HTTP://www.abaruchi.com/programacao/multiplicacao Matriz .html

[14] LAM-MPI, 1996. Disponível em: http://www.lam-mpi.org/

[15] Xen website, 2008. Acessado em 25/06/2008. Disponível em: http://www.cl.cam.ac.uk/research/srg/netos/xen/

[16] NAS Parallel Benchmark, 2007 Acessado em 25/06/2008. Disponível em: http://www.nas.nasa.gov/Resources/Software/npb.html

[17] Man Page time. Acessado em 25/06/2008. Disponível em: http://unixhelp.ed.ac.uk/CGI/man-cgi?time

[18] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, and M. Williamson, "Safe Hardware Access with the Xen virtua machine monitor", *Workshop on Operating System and Architecture Support for the on demand IT InfraStructure (OASIS)*, Oct 2004.

[19] Kim, Kangho Kim, Cheiyol Jung, Sung-In Shin, Hyun-Sup Kim, Jin-Soo, "Inter-Domain socket communication supporting high performance and full binary compatibility on Xen", *ACM/Usenix International Conference on Virtual Execution Environments*, ACM, New York, 2008, pp. 11-20.

[20] X. Zhang et. al., "XenSocket: A High-Throughput Interdomain Transport for Virtual Machines", *In Proc. Middleware*, 2007.

[21] Intel MPI Benchmark (IMB). Disponível em: http://www3.intel.com/cd/software/products/asmo-na/eng/cluster/219847.htm

[22] Credit Scheduler. Disponível em: http://wiki.xensource.com/xenwiki/CreditScheduler

[23] I.M. Leslie, D. Mcauley, R. Black, T. Roscoe, P.T. Barham, D. Evers, R. Fairbairns, and E. Hyden, "The Design and Implementation of na Operating System to Support Distributed Multimedia Applications", *IEEE Journal of Selected Areas in Communications*, 1996.

[26] Baruchi, Artur Midorikawa, Edson, "Influência do Programa de Escalonamento Credit Scheduler no Desempenho de Rede", *Workshop de Sistemas Operacionais*, SBC, Belem, Pará, Jul. 2008.

[27] Bucket Sort. Acessado em 25/06/2008. Disponível em: http://en.wikipedia.org/wiki/Bucket_sort

[28] W. Gropp, E. Lusk, N. Poss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. Parallel Computing, 1996, pp. 789-828.

[29] Site OpenMP. Acessado em 25/06/2008. Disponível em: http://www.openmp.org/specs/mp-documents/paper/paper.html