Virtualização – Problemas e desafios

Carlos Eduardo Seo[1] (008278)
IBM Linux Technology Center
Rod. Jornalista Fco. Aguirre Proença
13186-900 – Hortolândia, SP - Brasil
+55 (19) 2132-4339
eduseo@br.ibm.com

RESUMO

Nesse artigo são abordados os atuais problemas e desafios existentes na área de virtualização, em especial aqueles referentes à arquitetura de computadores. O objetivo desse trabalho é, primeiramente, oferecer uma abordagem geral sobre virtualização, conceitos, problemas e desafios. São abordados os temas: consolidação, *deployment*, *debugging*, virtualização em supercomputadores, gerenciamento de memória e gerenciamento de energia. Posteriormente, serão abordados em maior detalhe, dois casos relevantes à arquitetura de computadores: gerenciamento de memória baseado em redundância de dados [9] e o impacto de hierarquias de memória paravirtualizadas em *High Performance Computing* (HPC) [29]. O trabalho conclui que a área de virtualização possui terreno bastante fértil para pesquisa, principalmente nas áreas de sistemas operacionais e arquitetura de computadores.

Palavras-chave

Virtualização, *hypervisor*, desempenho, arquitetura, memória, supercomputadores, *cloud*, consolidação.

1. INTRODUÇÃO

Podemos definir virtualização como uma metodologia ou arcabouço que possibilita a divisão dos recursos de um computador entre múltiplos ambientes de execução [23]. O conceito surgiu nos anos 1960, no *IBM Thomas J. Watson Research Center*, durante o projeto M44/M44X, que visava avaliar os então recentes conceitos de compartilhamento de sistemas. Sua arquitetura era baseada em um conjunto de máquinas virtuais (VMs), uma para cada usuário. A máquina principal era um IBM 7044 (M44) e cada VM era uma réplica (imagem) experimental da 7044 (M44X). O espaço de endereçamento de cada M44X estava contido na hierarquia de memória da M44 e foi implementado através de técnicas de memória virtual e multiprogramação [8,18].

Hoje, em um sistema virtualizado, temos um software *host* sendo executado na máquina física, chamado *Virtual Machine Monitor* (VMM), ou *Hypervisor*. Esse software é responsável pela criação de ambientes simulados de computação, que são as VMs. Sistemas operacionais inteiros podem ser executados nessas máquinas virtuais como se estivessem sendo executados em um hardware real. O termo *Hypervisor* tem origem no sistema IBM VM/370, lançado em 1972, e se referia à interface de paravirtualização. No entanto, o *hypervisor* propriamente dito já existia desde o IBM CP-40, de 1967 [6]. Atualmente, temos dois tipos de *hypervisor*:

- Tipo 1 (*bare-metal*): executam diretamente no hardware do sistema *host*, atuando como controlador de hardware e monitor de sistemas operacionais *guest*. Ex: VMWare ESXi [25], IBM Power VM [12], Xen [28].
- Tipo 2 (*hosted*): executam em um sistema operacional comum e permitem a execução de sistemas operacionais *guest*. Ex: VMWare Workstation [27], Parallels Desktop [19], QEMU [21].

Desde o início da virtualização no final dos anos 1960, empresas como IBM [11], HP [10] e Sun [24] têm desenvolvido e vendido sistemas com suporte a virtualização. No entanto, o foco do mercado não era muito voltado para eles até o começo dos anos 2000. A evolução dos sistemas de hardware – aumento da capacidade de processamento, memória, disco – aliada a necessidade crescente de se fazer mais tarefas computacionais ao mesmo tempo com um custo cada vez menor fez com que a virtualização aparecesse em maior escala nos últimos anos.

Atualmente, não faltam motivações para o uso de virtualização: consolidação de carga de diversos servidores subutilizados em poucos ou apenas um servidor (consolidação de servidores), possibilidade de executar *legacy* software que não funcionam em hardware recente em VMs que simulem hardware compatível, VMs permitem a criação de ambientes seguros e isolados para execução de aplicações não-confiáveis, permite *debugging*/monitoramento de aplicações sem interferir no ambiente de produção, facilita migração de aplicações e servidores, permite simulação de hardware que o usuário não dispõe, entre outras [23].

Tais motivações trazem junto diversos desafios de pesquisa, tais como minimizar *overhead* de controle, gerenciar melhor o uso de memória, otimizar o consumo de energia em *datacenters*, facilitar gerenciamento e *deployment*, etc.

Esse artigo está estruturado da seguinte forma: na Seção 2 serão apresentadas algumas áreas de pesquisa que estão sendo exploradas atualmente no contexto de virtualização. Foram escolhidos alguns trabalhos apresentados em congressos e periódicos recentes para ilustrar cada área. A Seção 3, mostra em maior detalhe dois casos relevantes à arquitetura de computadores. E a Seção 4 apresenta as conclusões desse trabalho.

---

[1] Aluno de doutorado no Instituto de Computação, Universidade Estadual de Campinas, ceseo@ic.unicamp.br

2. PROBLEMAS E DESAFIOS

Nessa seção são apresentadas algumas áreas de pesquisa em virtualização, os problemas e desafios existentes. Para cada área, foi escolhido um trabalho publicado em periódicos/congressos recentes (2007-2009) como exemplo de pesquisa.

2.1 Consolidação de servidores

É praticamente impossível não associar virtualização e consolidação de servidores hoje em dia. O uso de máquinas virtuais para consolidação de sistemas em *datacenters* tem crescido rapidamente nos últimos anos devido às facilidades trazidas pela virtualização: facilidade de gerenciamento, disponibilização de máquinas, custo de infra-estrutura e consumo de energia [1]. Como conseqüência, várias empresas hoje oferecem produtos e serviços de virtualização, tais como VMWare, Microsoft [16], IBM and XenSource.

No entanto, consolidação traz uma conseqüência: as cargas das diversas VMs podem interferir em seus desempenhos. Com base nesse cenário, Apparao et al. [1], da Intel Corp., decidiram caracterizar e analisar o desempenho de uma carga representativa de um sistema consolidado.

O trabalho consiste na criação de um *benchmark* que represente uma carga típica de sistemas consolidados (*vConsolidate*). O estudo tem início com uma comparação de desempenho entre as aplicações sendo executadas em um ambiente isolado e depois em um servidor com múltiplas cargas, a fim de medir o impacto da consolidação em cada tipo de aplicação. São tiradas medidas como CPI e L2 cache MPI para tal comparação. É mostrado que qualquer aplicação sofre um impacto considerável em um ambiente consolidado (ver Figura 1). Para aplicações que fazem muito uso de CPU, a maior perda de desempenho é causada por interferências entre os caches e entre os cores. Os estudos mostram que tais aplicações se beneficiam de caches maiores. Já aplicações como *webservers* e *mail servers* se beneficiam mais com maior alocação de banda de rede.

São executados diversos experimentos para verificação de interferência entre cargas distintas, culminando na elaboração de um modelo de desempenho em servidores consolidados para auxiliar no projeto e antecipação do desempenho de cargas virtualizadas em plataformas futuras, levando em conta os aspectos medidos no trabalho.

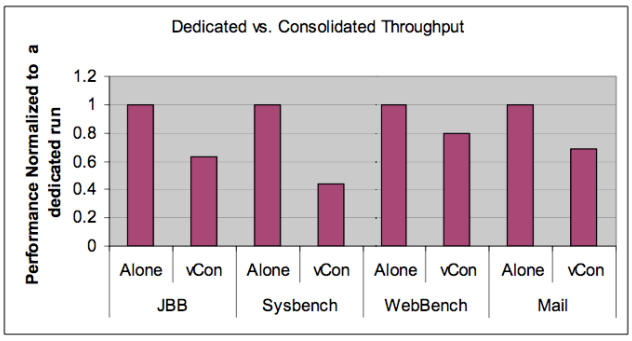

Figura 1: Impacto da consolidação no *throughput* de aplicações [1]

Como trabalhos futuros, os autores citam a necessidade de medir outros *overheads* trazidos pela virtualização: *context switches*, interrupções e *page faults*, visto que o custo (em desempenho) dessas operações muda em um ambiente virtualizado. Além disso, deve ser tratada também a questão da escalabilidade, para o caso da existência de mais de uma *Consolidated Stack Unit* (CSU).

Essa é uma área bastante fértil para a pesquisa dentro de virtualização, pois existem poucos trabalhos e falta muito ainda para que a questão de desempenho em servidores consolidados esteja madura.

2.2. *Deployment*

Em ambientes com várias cargas de trabalho, é interessante o fato de uma máquina virtual possa ser criada e estar pronta para uso o mais rápido possível. Isso se torna mais crítico com a tendência atual de migrar os custos de manutenção e gerenciamento de *datacenters* para terceiros através do *Cloud Computing*. Esse modelo tem a virtualização como um de seus principais vetores, que possibilita a criação (*deployment*) de diversas VMs conforme a demanda dos clientes. No entanto, a maior parte das APIs de *cloud computing* não consegue fazer o deployment das máquinas de forma ágil. Nesse contexto, Lagar-Cavilla et al. [13] propõem uma solução para *deployment* de VMs em menos de 1 segundo chamada *SnowFlock*.

O trabalho se baseia no conceito de *fork* aplicado em VMs. O funcionamento é basicamente igual ao *fork* de processos: uma VM pai faz uma chamada de *fork* que cria um certo número de VMs filhas. Cada VM filha tem um sistema idêntico (inclusive estado) ao da VM pai e também um identificador único (*vmid*). Após o *fork*, o sistema operacional e o disco virtual de cada VM é independente das outras, assim como seus estados. As VMs filhas possuem natureza efêmera, ou seja, uma vez destruídas, sua memória e disco virtual são descartados também. Qualquer alteração que se deseje passar para a VM pai (ou outras VM filhas), deve ser transmitida explicitamente.

O *fork* em tempo abaixo de 1 segundo é conseguido através de técnicas de *lazy state replication*: apenas um pequeno arquivo (*VM descriptor*) é utilizado para criação e inicialização da VM filha. No decorrer da execução da VM filha, um mecanismo copia o estado da memória a partir da VM pai *on-demand*. Algumas modificações são feitas no *kernel* para que as páginas de memória que serão sobrescritas não sejam transferidas durante o *fork*, minimizando o uso de banda de rede. Para se ter uma idéia, o *fork* de um *footprint* de 1GB custa apenas 40MB (ver Tabela 1). Outro mecanismo usado é o *multicast* do estado das VMs, devido a localidade dos acessos à memória pelas VMs. Isso permite o *fork* de diversas VMs a um custo praticamente igual ao do *fork* de uma única VM.

A solução é testada utilizando aplicações típicas de *cloud*: bioinformática, *rendering*, compilação paralela e serviços financeiros. Para cada aplicação, são criadas 128 *threads* de execução: 32 VMs com 4 *cores* SMP distribuídas em 32 *hosts* diferentes. Os resultados mostram que o *SnowFlock* é capaz de instanciar dezenas de VMs em *hosts* diferentes em tempos abaixo de 1 segundo, com um baixo *overhead* de tempo de execução e baixo uso dos recursos de IO do *cloud* (ver Tabela 1), um ganho significativo em relação a solução adotada pelo Xen.

Os autores afirmam que ainda existe espaço para novas pesquisas que explorem o *fork* para VMs. Um exemplo seria o uso dessa técnica juntamente com APIs de paralelismo de dados, como o *MapReduce* [7], ou ainda em situações de migração em massa de VMs entre geografias diferentes.

Tabela 1. *SnowFlock* vs. *Suspend/Resume* (Xen) [13]

Técnica	Tempo* (s)	Estado (MB)
SnowFlock	70,63 ± 0,68	41,79 ± 0,7
S/R *multicast*	157,29 ± 0,97	1124
S/R sobre NFS	412,29 ± 11,51	1124

2.3. Gerenciamento de energia

Minimizar os gastos com energia tem sido cada vez mais uma preocupação entre os administradores de *datacenters*. Ao mesmo tempo, a virtualização está cada vez mais sendo adotada nesse ambiente. Dentro desse contexto, Nathuji et al. [17] propõem um novo mecanismo de gerenciamento de energia chamado *VirtualPower Management* (VPM). Trata-se de uma extensão ao Xen que permite um gerenciamento mais eficiente do consumo de energia entre VMs, com impacto mínimo no desempenho das aplicações.

O VPM se baseia basicamente em 2 pontos: (1) oferecer as VMs um conjunto maior de estados possíveis de energia (*soft states*), que são acessíveis as políticas específicas de cada aplicação em cada VM e (2) o uso das mudanças de estado requisitadas pelas VMs como entrada para políticas de gerenciamento de energia em nível de virtualização (mais próximo ao *hardware*). No Xen, toda a inteligência do sistema fica em *Domain0*, o que possibilita a implementação do VPM sem modificações pesadas no *hypervisor*.

Foram testados dois tipos de carga de trabalho: uma transacional e uma de serviços *web*. Em ambos os casos foi verificado um gerenciamento de energia mais ativo, tendo como conseqüência direta o menor consumo de energia, mas sem afetar negativamente o desempenho das aplicações. Em alguns casos, a economia chegou até 34%.

Essa é uma área de pesquisa que, embora tenha muitos trabalhos publicados, sempre tem espaço, visto que nos dias de hoje, melhoras no consumo de energia são muito bem vistas pelo mercado.

2.4. *Debugging*/monitoramento de aplicações

A análise de programas em tempo de execução tem várias aplicações, desde segurança até *debugging*. No entanto, o *overhead* de análise impacta significativamente no desempenho do ambiente de produção. Como uma alternativa viável para solução desse problema, os pesquisadores Chow et al. [5] , da VMWare, formularam uma solução baseada em VMs que permite a realização de análises pesadas em ambiente de produção chamada de *Aftersight*.

Para que isso seja possível, a análise é desacoplada da carga de trabalho – as entradas não-determinísticas da VM de produção são registradas e é feito um *replay* da carga em uma VM diferentes. Com isso, pode-se ter uma análise em tempo real *inline*, *best-effort*, ou ainda, *offline*. Existe ainda a possibilidade de se executar múltiplas análises para uma mesma carga ou incluir novas análises de acordo com a necessidade.

O trabalho é extremamente denso para ser apresentado completamente nesse texto, porém, dos resultados apresentados, dois chamam a atenção. Primeiramente, é mostrado o impacto de uma análise *inline* usando o *Aftersight*. A análise utilizada simula uma execução de antivírus durante uma operação de uso intensivo de disco. Os resultados mostram que o desempenho é cerca de 12% melhor se compararmos com uma análise *inline* tradicional. Outro teste é uma análise pesada sendo executada em paralelo com a carga (*best-effort*). A análise escolhida nesse caso é uma verificação de condição de escrita em um mapa de memória durante uma compilação de *kernel* Linux. Os resultados mostram que o *Aftersight* apresenta desempenho 2x melhor do que o de uma análise *inline* tradicional nesse caso.

Os autores concluem que a análise dinâmica da execução de programas é uma técnica bastante promissora para a solução de diversos problemas, mas que é limitada pelo impacto no desempenho das cargas de trabalho. No entanto, a adoção da virtualização, técnicas como a análise desacoplada mostrada no trabalho se mostram alternativas bastante promissoras.

2.5. Gerenciamento de memória

Um dos grandes desafios em virtualização é o compartilhamento eficiente de memória entre as VMs, que é um dos maiores gargalos na consolidação de sistemas. Pesquisas mostram que é possível melhorar o consumo de memória através do compartilhamento de páginas entre VMs que executam sistemas operacionais e aplicativos similares. Gupta et al. [9] mostram que além disso, duas outras técnicas podem ser utilizadas para se obter melhor uso da memória entre VMs: *patching* de páginas e compressão de páginas. Com base nisso, eles implementam o *Difference Engine*, uma extensão ao VMM do Xen e mostram que os ganhos são bastante significativos (até 90% para cargas similares e até 65% para cargas heterogêneas).

A implementação e análise dessa técnica será descrita em detalhe na seção 3.1.

2.6. Virtualização em supercomputadores

O uso de técnicas de virtualização em supercomputadores ainda é incipiente. Primeiro, porque essas máquinas foram construídas para serem utilizadas em tarefas bastante específicas; segundo, o seu alto custo teoricamente as torna menos competitivas que soluções para sistemas *commodity*. No entanto, existe um projeto em andamento no IBM Research chamado *Project Kittyhawk* [20] que explora a construção e os impactos de um computador de escala global capaz de abrigar a toda a internet como uma aplicação. O trabalho de Appavoo et al. [2] apresenta uma visão geral desse projeto, motivação, descrição do firmware e sistema operacional utilizado e os primeiros resultados da experiência.

* Tempo de *deployment* das VMs somado com o tempo de execução do benchmark SHRiMP [22].

Os autores enfatizam que ambos os modelos existentes hoje – *clusters* e sistemas multiprocessados com memória compartilhada – não são adequados para um computador de escala global. É proposto que um híbrido dos dois modelos seria a melhor solução. Um sistema que tenha um processo de fabricação em larga escala já consolidado; empacotamento, alimentação, refrigeração e instalação eficientes; arquitetura de nós simples e minimalista; barramento de interconexão do tipo NUMA escalável; domínios de comunicação configuráveis e impostos pelo *hardware*; e *software* de gerenciamento e controle escalável. Nesse contexto, o Blue Gene/P se mostra uma plataforma ideal para a realização do modelo. A plataforma é descrita a seguir.

Cada nó do Blue Gene/P possui 4 *cores* PowerPC 450 de 850 MHz e 2 GB de RAM. Os nós são agrupados em placas com 32 nós cada e as placas são agrupadas em outro conjunto de 16 placas cada. Cada *rack* do Blue Gene/P tem 2 conjuntos de 16 placas, totalizando 1024 processadores e 2 TB de RAM. Os *racks* podem ser agrupados para constituir uma única instalação, até o limite de 16384 *racks*, o que resulta em um sistema com 67,1 milhões de *cores* e 32 PB de RAM. Apesar desses números monstruosos, é importante notar que cada nó pode ser visto como um computador de uso geral, com processadores, memória e unidades de I/O.

Para explorar as possibilidades da máquina, foi utilizado um *boot loader* (U-Boot) modificado, o L4 Hypervisor [15] e Linux como sistema operacional das VMs. Foram escolhidas diversas aplicações dentro do contexto de web 2.0 para a realização dos experimentos.

Os resultados mostram que é factível o uso do Blue Gene/P como uma plataforma para serviços web. As redes de alta velocidade da plataforma garantem conectividade interna e externa, a alta capacidade de memória permite oferecer *storage* de rede de baixa latência e é possível agrupar diversos processadores em redes colaborativas. Além disso, apresenta rápida escalabilidade (mais rápido do que expandir um *datacenter* comum) e bastante flexibilidade. Além disso, a solução pode ser atrativa em termos de custo também. Servidores *commodity* podem ser baratos individualmente, mas um *cluster* é caro de se comprar e manter (energia e refrigeração). Outro ponto são as conexões de rede: apesar de termos placas muito baratas, a infra-estrutura de *switching* necessária em grandes *clusters* é cara e o preço não escala linearmente com o número de portas. Isso faz com que a solução apresentada seja uma ordem de magnitude mais eficiente (em termos de valores de compra e operação) do que um *cluster* tradicional para uma grande gama de cargas de trabalho *web*.

Tais resultados abrem portas para mais um ramo de pesquisa na área de virtualização, com oportunidades principalmente em nas áreas de sistemas operacionais e arquitetura de computadores.

3. ANÁLISE

3.1. Gerenciamento de memória baseado em redundância de dados

Conforme foi mencionado na seção 2.4, Gupta et al. [9] desenvolveram um mecanismo de gerenciamento de memória no VMM do Xen que possibilita o melhor uso da memória compartilhada entre diversas VMs, através da aplicação de 3 técnicas: compartilhamento de páginas, *patching* de páginas e compressão de páginas.

O compartilhamento foi a primeira técnica utilizada em virtualização. O VMWare ESX Server já fazia o uso de compartilhamento de páginas entre VMs diferentes. No entanto, para que o compartilhamento seja possível, as páginas compartilhadas devem ser idênticas. Porém, é possível tirar proveito de páginas quase idênticas também. Nesse contexto, aparece a técnica de *patching*: páginas quase idênticas podem ter sua parte idêntica compartilhada e *patches* são aplicados para reconstruir a página desejada. Por último, quando as páginas têm probabilidade de não serem usadas num futuro próximo, elas são comprimidas. A Figura 2 mostra um esquemático das 3 técnicas.

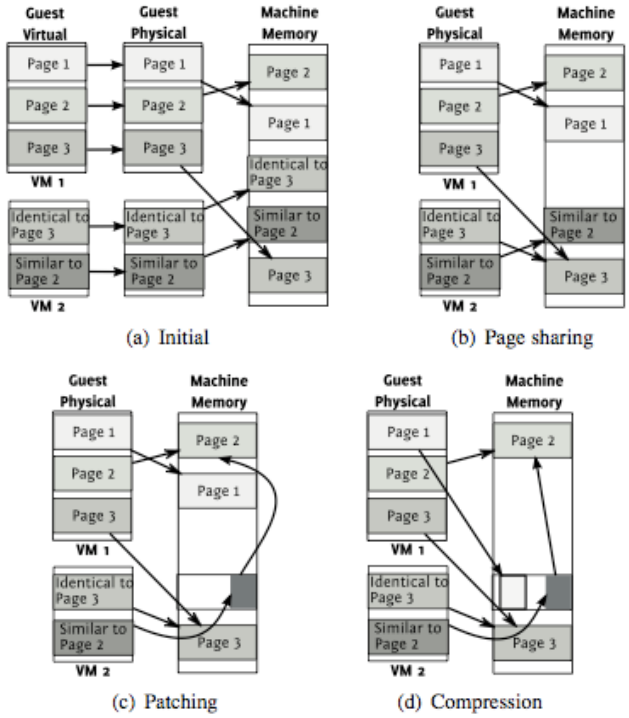

Figura 2: Exemplo mostrando as 3 técnicas utilizadas pelo *Difference Engine*. No final, temos um ganho aproximado de 50% em espaço em relação ao armazenamento original [9].

O compartilhamento é implementado da mesma forma em que trabalhos anteriores: é feita uma varredura na memória e um *hash* de cada página. As páginas são indexadas com base em seu valor de *hash*. Páginas idênticas apresentam o mesmo valor de *hash* e quando uma colisão é detectada, tem-se um potencial candidato ao compartilhamento. É feita então uma varredura *byte* a *byte* para se ter certeza que as páginas são realmente idênticas e, em caso afirmativo, elas são compartilhadas e a memória virtual passa a apontar para uma delas. Páginas compartilhadas são marcadas como somente leitura, de tal forma que, quando VM tenta escrever, uma *page fault* ocorre. O VMM então retorna uma cópia privada da página para a VM que provocou a falha e atualiza os mapas de memória. Se em algum momento nenhuma VM referencia uma página compartilhada, ela é liberada.

Para a operação de *patching*, o *Difference Engine* utiliza um mecanismo de *hash* para encontrar possíveis candidatos. Cada página é indexada através de *hashes* de dois blocos de 64 KB em posições fixas da páginas (a escolha inicial dessas posições é aleatória). Ou seja, cada página tem 2 índices na tabela *hash*. Para que se encontre uma página candidata ao *patching*, o mecanismo

calcula *hashes* para blocos nas mesmas posições da página e procura na tabela a melhor página entre as (no máximo duas) encontradas. Como esse mecanismo possui um *overhead* não desprezível, só é aplicado a páginas que são utilizadas com pouca freqüência.

Já para a compressão, ela só é aplicada quando a taxa de compressão é alta e se as páginas são utilizadas com pouca freqüência, caso contrário, o *overhead* da operação ultrapassa os benefícios trazidos pela técnica. Páginas são identificadas através de um algoritmo que implementa um relógio global para identificar a freqüência de acesso. As páginas comprimidas são invalidadas e movidas para uma área de armazenamento na memória da máquina física. Caso um acesso seja feito, o mecanismo faz a descompressão e retorna a página para a VM requisitante.

Além disso, o *Difference Engine* também identifica páginas candidatas a *swapping* dentre aquelas que foram marcadas para *patching* e compressão. No entanto, a decisão de quanto e quando fazer *swap* para disco fica a cargo da aplicação em *userspace*.

Tudo isso faz com que o gerenciamento de memória seja muito mais eficiente se usamos o Xen + *Difference Engine* do que o VMWare ESX Server, conforme mostram os resultados. Foram utilizados vários cenários, mas para fins de análise, será mostrado aqui apenas um deles, que é bastante significativo pois utiliza uma carga de trabalho do mundo real. São 3 VMs completamente diferentes: Windows XP SP1 executando RUBiS, Debian Linux 3.1 compilando o kernel Linux e Slackware Linux 10.2 compilando o vim-7.0 e executando o *benchmark* lmbench logo em seguida. Se observarmos a Figura 3, veremos que o *Difference Engine* consegue economizar até 45% mais memória que o VMWare ESX Server (que utiliza apenas compartilhamento).

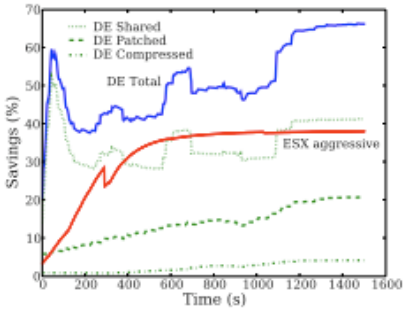

Figura 3: Economia de memória para a carga heterogênea[9].

Nota-se que se usarmos apenas o compartilhamento, o desempenho em geral é similar entre os dois VMMs. No entanto, ao adicionarmos *patching* e compressão, o *Difference Engine* dá um salto grande em desempenho, mostrando que essas técnicas também são importantes no gerenciamento de memória no VMM, pois o *overhead* medido foi de apenas *7%*. Os outros resultados mostram que se obtém uma economia de até 90% em cargas homogêneas e de até 65% em cargas heterogêneas.

Como trabalhos futuros, os autores citam que é possível explorar a utilização do mecanismo do *Difference Engine* para melhorar o desempenho de um sistema operacional convencional isoladamente também.

3.2. Impacto da hierarquia de memória paravirtualizada em HPC

A virtualização foi ignorada por um bom tempo pelos usuários de aplicações de computação intensiva devido ao potencial decremento no desempenho das aplicações. No entanto, para esses usuários, a paravirtualização se mostra uma alternativa interessante. Na paravirtualização, os sistemas operacionais *host* e *guest* são modificados para oferecer o melhor desempenho possível quando é utilizada a virtualização. Vários estudos [29] mostraram que para aplicações de HPC, os sistemas paravirtualizados apresentam desempenho muito próximo do sistema nativo. No entanto, informações a respeito do desempenho em situações nas quais a memória é escassa são raras. Isso é importante porque o desempenho de várias aplicações de álgebra linear depende das características do uso da memória. Nesse contexto, Youseff et al. [30] mostram um estudo detalhado do impacto da hierarquia de memória paravirtualizada em aplicações de HPC. Mais especificamente, deseja-se saber como que a paravirtualização afeta o *autotuning* das aplicações e como ela afeta o desempenho das aplicações. Para isso, será utilizada uma das ferramentas de *autotuning* mais populares que existem, o ATLAS (*Automatically Tuned Linear Algebra Software*) [3], para fazer o *autotuning* de duas bibliotecas muito usadas em álgebra linear: BLAS [4] e LAPACK [14]. Como solução de virtualização, foi utilizado o Xen.

Foram preparados 3 sistemas: um nativo, uma VM com privilégios (*Dom0*, na notação do Xen) e uma VM sem privilégios (*DomU*), todos com Linux 2.6.16. Duas configurações de memória foram adotadas: 256 MB e 756 MB.

Os primeiros experimentos foram feitos visando observar o impacto da paravirtualização no *autotuning*. Os resultados mostram que a paravirtualização não influi na caracterização do sistema (detecção de hardware idêntico para os 3 sistemas), nem impõe *overhead* de desempenho em operações entre registradores de ponto flutuante. No teste de configuração de cache, no qual o ATLAS tenta encontrar o tamanho ótimo do bloco de cache (tanto para L1 quanto para L2) a ser usado em operações de multiplicação entre matrizes, os resultados mostram que o ATLAS não vê diferença entre os sistemas na hora de escolher o tamanho ótimo do bloco para cache L2.

O segundo conjunto de experimentos investiga o impacto da paravirtualização em diferentes níveis da hierarquia de memória durante a execução de aplicações HPC de uso intensivo de memória. Foi utilizado um código de multiplicação entre matrizes com precisão dupla que possui consumo de memória crescente até 350 MB e o desempenho foi medido em MFLOPS.

Os resultados mostram que a paravirtualização tem impacto muito pouco significativo no desempenho desse tipo de aplicação, o que é um boa surpresa, pois a paravirtualização introduz mais um nível de escalonamento de processos e indireção de I/O, o que poderia causar facilmente um aumento no TLB *miss rate*, por exemplo. O que temos é um perfil de hierarquia de memória muito similar entre os 3 sistemas, desde o acesso ao cache, até o *swap* em disco.

Tais resultados mostram que aplicações de HPC podem tirar proveito de estruturas de *clusters* virtuais e *cloud computing*, visto que a virtualização não influencia no desempenho dessas aplicações.

4. CONCLUSÕES

Esse trabalho apresentou uma visão geral da área de virtualização, mostrando quais são as áreas de pesquisa sendo exploradas atualmente e o que ainda existe de oportunidade para novos trabalhos. Para as áreas de sistemas operacionais e arquitetura de computadores, existem ainda vários tópicos de pesquisa que valem a pena ser explorados, principalmente em gerenciamento de memória e o uso de supercomputadores como base de VMs.

5. AGRADECIMENTOS

Agradecimentos ao Prof. Dr. Rodolfo Jardim de Azevedo (IC-UNICAMP) e à Dra. Dilma da Silva (IBM Thomas J. Watson Research Center) pela orientação nesse trabalho de pesquisa.

6. REFERÊNCIAS

[1] Apparao, P.; Iyer, R.; Zhang, X.; Newell, D. & Adelmeyer, T., Characterization & analysis of a server consolidation benchmark, *VEE '08: Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on virtual execution environments*. ACM, **2008**, pp. 21-30.

[2] Appavoo, J.; Uhlig, V. & Waterland, A., Project Kittyhawk: building a global scale computer: Blue Gene/P as a generic computing platform, *SIGOPS Oper. Syst. Rev*. ACM, **2008**, Vol. 42 (1), pp. 77-84.

[3] ATLAS
http://math-atlas.sourceforge.net
24/06/2009.

[4] BLAS
http://www.netlib.org/blas
24/06/2009.

[5] Chow, J.; Garfinkel, T. & Chen, P.M., Decoupling dynamic program analysis from execution in virtual environments, *ATC '08: USENIX 2008 Annual Technical Conference*. USENIX Association, **2008**, pp. 1-14.

[6] Creasy, R.J., The origin of the VM/370 time-sharing system, *IBM Journal of Research & Development Vol. 25, No. 5*. IBM, **1981**, pp. 483-490.

[7] Dean, J. & Ghemawat, S., MapReduce: Simplified Data Processing on Large Clusters, *Commun. ACM*. ACM, **2008**, Vol. 51 (1), pp. 107-113.

[8] Denning, P., ACM president's letter: performance analysis: experimental computer science as its best, *Commun. ACM*. ACM, **1981**, Vol. 24 (11), pp. 725-727.

[9] Gupta, D.; Lee, S.; Vrable, M.; Savage, S.; Snoeren, A. C.; Varghese, G.; Voelker, G. M. & Vahdat, A., Difference Engine: Harnessing Memory Redundancy in Virtual Machines, *OSDI'08: Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation. USENIX Association,* **2008**, pp. 309-322.

[10] HP Co.
http://www.hp.com
12/06/2009.

[11] IBM Corp.
http://www.ibm.com
12/06/2009.

[12] IBM PowerVM
http://www-03.ibm.com/systems/power/software/virtualization
12/06/2009.

[13] Lagar-Cavilla, H.A.; Whitney, J.A.; Scannell, A.M.; Patchin, P.; Rumble, S.M.; de Lara, E.; Brudno, M. & Satyanarayanan, M., SnowFlock: rapid virtual machine cloning for cloud computing, *EuroSys '09: Proceedings of the fourth ACM European conference on computer systems*. ACM, **2009**, pp. 1-12.

[14] LAPACK
http://www.netlib.org/lapack
24/06/2009.

[15] Liedtke, J., On µ-kernel construction, *SOSP '95: Proceedings of the 15th ACM Symposium on Operating System Principles*. ACM, **1995**, pp. 237-250.

[16] Microsoft Corp.
http://www.microsoft.com
12/06/2009.

[17] Nathuji, R. & Schwan, K., VirtualPower: coordinated Power management in virtualized enterprise systems, SOSP '07: Proceedings of twenty-first *ACM SIGOPS symposium on operating systems principles*. ACM, **2007**, pp. 265-278.

[18] O'Neill, W., Experience using a time sharing multiprogramming system with dynamic address relocation hardware, *Proc. AFIPS Computer Conference 30*. SJCC, **1967**, pp. 78-117.

[19] Parallels Desktop
http://www.parallels.com/products/desktop
12/06/2009.

[20] Project Kittyhawk
http://domino.research.ibm.com/comm/research_projects.nsf/pages/kittyhawk.index.html
23/06/2009.

[21] QEMU
http://www.nongnu.org/qemu
12/06/2009.

[22] SHRiMP
http://compbio.cs.toronto.edu/shrimp
17/06/2009.

[23] Singh, A., An Introduction To Virtualization.
http://www.kernelthreads.com/publications/virtualization
12/06/2009.

[24] Sun Microsystems Inc.
http://www.sun.com
12/06/2009.

[25] VMWare ESXi
http://www.vmware.com/products/esxi
12/06/2009.

[26] VMWare Inc.
http://www.vmware.com
12/06/2009.

[27] VMWare Workstation
http://www.vmware.com/products/ws
12/06/2009.

[28] Xen.org
http://www.xen.org
12/06/2009.

[29] Youseff, L.; Wolski, R.; Gorda, B. & Krintz, C., Evaluating the Performance Impact of Xen on MPI and Process Execution For HPC Systems, *VTDC '06: Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing*. IEEE, **2006**, pp. 1.

[30] Youseff, L., Seymour, K., You, H., Dongarra, J. & Wolski, R., The impact of paravirtualized memory hierarchy on linear algebra computational kernels and software, *HPDC '08: Proceedings of the 17th international symposium on High performance distributed computing*. ACM, **2008**, pp. 141-152.