# Elasticidade de Memória em Máquinas Virtuais Utilizando Média Móvel Exponencial

**Artur Baruchi, Edson Toshimi Midorikawa**

Laboratório de Arquitetura e Computação de Alto Desempenho

Departamento de Engenharia de Computação e Sistemas Digitais – Escola Politécnica da Universidade de São Paulo – São Paulo – Brasil

{artur.baruchi,edson.midorikawa}@poli.usp.br

***Abstract.*** *The main advantage when using virtualization technology is to improve resource usage. Besides that, virtualization is the main technology involved in cloud computing. This paradigm's goal is the resource sharing however sharing resources, isn't trivial and, for example, adding or removing memory should be done carefully. Our current work presents a mechanism for virtual machines dynamic memory allocation based on exponential moving average. By means of experiences with two different benchmarks, it was observed that using the proposal, the VMs were able to adapt and support high memory demand fluctuation and, in some cases, with low performance degradation of 3%.*

***Resumo.*** *Uma das principais vantagens da virtualização é a melhor utilização de recursos computacionais. Além do grande atrativo da virtualização em melhorar o uso de recursos computacionais a virtualização é uma das principais tecnologias envolvidas na computação em nuvem. Um dos objetivos deste paradigma é o compartilhamento de recursos computacionais, entretanto o compartilhamento de alguns recursos, não é trivial e exige maior cuidado como o ato de remover ou adicionar certa quantidade de memória. Este trabalho apresenta uma técnica para a alocação de memória dinâmica em máquinas virtuais utilizando média móvel exponencial. Por meio de experimentos foi possível observar que através da solução proposta as MVs puderam se adaptar as demandas de memória e em alguns casos com degradação de apenas 3% no desempenho.*

## 1. Introdução

Apesar do conceito de virtualização datar do início dos anos 70 [Goldberg, 1974], esta começou a se popularizar há pouco tempo. Ironicamente, os motivos do ressurgimento da virtualização são, basicamente, os mesmos motivos pelos quais esta tecnologia entrou em desuso nos anos 80 e 90 [Rosenblum, 2005]. O ressurgimento tem como principal motivo o barateamento e aumento do poder computacional de componentes de *Hardware*, como memória e processador. Os processadores *multi-core* e máquinas com diversos *gigabytes* de memória RAM deixaram de ser privilégio de grandes empresas e universidades e começaram a se tornar comuns, tanto em empresas menores como em computadores pessoais.

Sistemas que antes eram executados em máquinas *monoprocessadas* e com alguns *megabytes* de memória começaram a ser migrados para servidores mais potentes.

Com a ocorrência destes eventos, os computadores começaram a ficar cada vez mais ociosos, implicando em desperdício de ciclos de processador e espaço em memória. Diante deste cenário, surgiu a necessidade do desenvolvimento de alguma tecnologia capaz de aproveitar os equipamentos subutilizados. Assim a virtualização começou a ressurgir a partir de então como uma das soluções para melhorar o aproveitamento dos recursos.

A virtualização rapidamente se mostrou de grande utilidade em diversos campos da computação devido às características das Máquinas Virtuais (MV). Uma das áreas que mais se beneficiaram destas características foi a computação em grade (*grid computing*) ([Muntean, 1994], [Mirtchovski, 2007]). Algumas funcionalidades, como a facilidade de se migrar uma MV instalada em uma determinada máquina física para outra com maior disponibilidade de recursos e de forma transparente, tornaram a virtualização uma ferramenta de grande utilidade neste campo da computação.

Além de características atraentes para outras áreas da computação já consolidadas, a virtualização viabilizou antigas idéias como a utilização da computação como serviço [Parkhill, 1966]. A idéia do uso da computação como serviço deu origem ao termo computação em nuvem, que vem sendo amplamente discutida e estudada atualmente ([Kandukuri, 2009], [Vecchiola, 2009]). O papel da virtualização na computação em nuvem é tão importante que em algumas definições deste paradigma é utilizado o termo virtualização [Buyya et al, 2008]:

> *"... tipo de sistema paralelo e distribuído que consiste em uma coleção de computadores* ***virtualizados*** *e interconectados que são dinamicamente fornecidos e apresentados como um ou mais recursos computacionais unificados fundamentados em um acordo de nível de serviço..."*

A elasticidade é um termo que vem sendo bastante utilizado devido à grande difusão da computação em nuvem. O presente trabalho aborda a elasticidade de memória em Máquinas Virtuais. E pode ser entendida como a capacidade de um ambiente de se adaptar a carga de trabalho. Esta capacidade de adaptação à demanda de recursos é algo complexo, os principais fatores que dificultam a criação de um mecanismo que seja capaz de se adaptar as demandas são:

- Identificação do recurso escasso;
- Quantificar a demanda do recurso para satisfazer as necessidades do sistema;
- Identificação do momento em que o recurso alocado pode ser removido, sem prejudicar o sistema;

Para tratar os problemas citados acima, foi implementado no Monitor de Máquinas Virtuais (MMV) Xen [Barhan et al., 2003] um mecanismo que utiliza o cálculo da Média Móvel Exponencial (MME) para identificar as necessidades da MV e fazer a alocação de recurso conforme a demanda. Em alguns experimentos realizados foi possível observar que a MV obteve boa elasticidade de memória com pouca degradação de desempenho (apenas 3%).

Na próxima seção serão abordados alguns trabalhos cujo foco é o gerenciamento de recursos em MVs. Na seção 3, serão discutidos alguns aspectos da Elasticidade de Memória, como as principais dificuldades encontradas e os seus benefícios. A estratégia implementada neste trabalho é apresentada na seção 4 e em seguida, na seção 5, os

resultados obtidos através da execução de *benchmarks* serão apresentados. Por fim, as conclusões e trabalhos futuros são discutidos na seção 6.

## 2. Trabalhos Relacionados

Alguns trabalhos vêm sendo realizados com o objetivo de prover um ambiente virtualizado mais flexível. O primeiro trabalho abordado [Moskovsky, 2008] apresenta um método para encontrar a melhor configuração para uma MV baseado em um *Service Level Agreement* (SLA). Os autores apresentam um *Framework* que através de um mecanismo de monitoração e no SLA de um determinado serviço, o *Framework* realiza alterações nos recursos alocados das MVs para que a SLA seja satisfeita sem desperdício de recursos.

Para que o *Framework* fosse implementado, os autores realizaram um estudo sobre a influência de alguns parâmetros de configuração das MVs no desempenho de *benchmarks* com diferentes comportamentos. Esse estudo teve como principal objetivo desenvolver um *Profile* para que o *Framework* pudesse alterar o parâmetro correto, isto é, que realmente influenciará no desempenho de determinada carga de trabalho. Por exemplo, foi identificado que o aumento de memória em MVs que estavam executando uma aplicação *Web* (aplicação de publicação de dados espaciais via *Web*) não sofre influencias acentuadas em alterações na quantidade de memória, entretanto alterando-se a quantidade de tempo de processamento das MVs (aumento de *time slice*) o tempo de resposta obteve melhoras significativas. Ao submeter as MVs a diferentes cargas de trabalhos e SLAs, os autores observaram que o término do processamento de *Jobs* sofreu desvio de apenas 0,5% do tempo acordado, portanto um *Job* que demoraria 40 minutos para processar terminou com atraso de 10 a 15 segundos.

Outro trabalho que aborda a alocação de recursos em um ambiente composto por MVs [Bertogna et. al. 2009] foi realizado com o principal objetivo de otimizar o uso de recursos computacionais em uma grade de computadores (*Grid*). No trabalho os autores elaboraram um algoritmo que encontra o melhor conjunto de *clusters* que fosse capaz de atingir o melhor desempenho de uma determinada tarefa. Para que isso seja possível, o algoritmo analisa o que é mais custoso para o ambiente, se é a migração da MV para outra máquina física ou se há a possibilidade de readequar os recursos alocados.

Os autores utilizam um algoritmo de maximização, conhecido como *Hill-Climbing*, para encontrar as melhores máquinas que compõe o *Grid* e em seguida escolher as máquinas mais próximas, com o objetivo de diminuir o custo de comunicação entre elas (por meio do algoritmo de Kruskal). Após a implementação deste escalonador, os autores atingiram uma melhora de 20% no tempo do processamento e com 100% de utilização das máquinas escolhidas para realizar o processamento da tarefa.

Algumas técnicas novas de gerência de recursos têm sido propostas, como pode ser observado no trabalho feito em [Arcangeli, 2009]. O *Kernel Shared Memory* (KSM) tem como objetivo compartilhar páginas de memória com o mesmo conteúdo, não só de MVs criadas por meio do KVM [Kivity et. al., 2007], mas também compartilhar memória entre os processos. O KSM foi disponibilizado na versão 2.6.31 do *Kernel* do Linux. Apesar do KSM não ser uma técnica que viabiliza a elasticidade de memória, é uma técnica que otimiza o uso da memória, causando menos desperdícios.

## 3. Elasticidade de Memória

A elasticidade de recursos é um termo que vem sendo bastante utilizado por conta da popularização da computação em nuvem [Armbrust et. al. 2009]. A idéia principal da elasticidade é  criar um ambiente que se adapte às diferentes cargas de trabalho impostas ao sistema. Em geral, uma máquina real tradicional tem os recursos alocados de forma estática e qualquer variação na alocação (adição ou remoção) implica em *downtime* do ambiente. Com o ressurgimento da virtualização a alocação se tornou um pouco mais flexivel, principalmente quando o recurso em questão é o processador.

Na alocação estática de recursos ocorrem dois problemas (Figura 1) podem ser a falta ou o desperdicio de recurso. O desperdício de recurso se dá quando o ambiente foi provisionado para suportar picos de utilização, mas que ocorrem esporadicamente. Já a falta de recursos acontece quando o ambiente foi projetado somente para satisfazer a demanda do dia-a-dia e em momentos de picos de utilização este poderá sofrer lentidões ou até mesmo indisponibilidades.

Uma das principais dificuldades na otimização da alocação de recursos é manter o ambiente com um desempenho aceitável. Muitas vezes o próprio ato de alocar ou remover um determinado recurso pode causar sobrecarga (como no caso da memória que é utilizado um mecanismo chamado *Balloon Driver* [Waldspurger, 2002] que estimula os algoritmos de paginação do Sistema Operacional instalado na MV a procurar páginas de memória descartáveis).

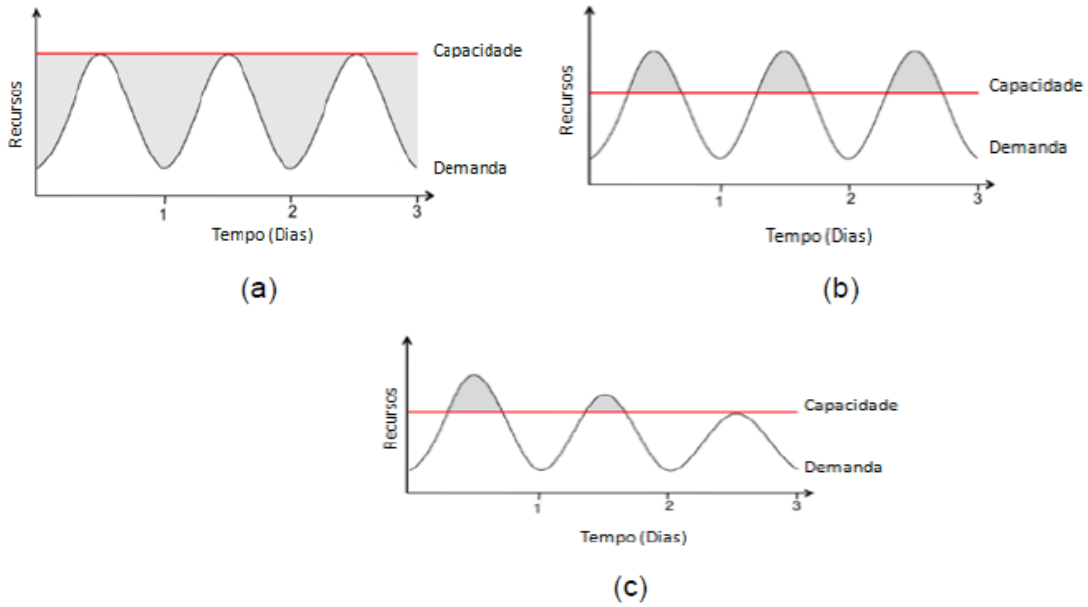



**Figura 1. (a) Provisão para picos. (b) e (c) Provisão para demanda média [Armbrust et. al. 2009]**

Este trabalho tem como foco principal a implementação da elasticidade na memória. Em geral, pesquisas que abordam a elasticidade de recursos tratam apenas do processador e a adaptação da memória para a carga de trabalho é realizada muitas vezes de forma empírica ou de forma proporcional à alocação do processador. Os principais problemas que envolvem a implementação de elasticidade de memória são:

- *Identificação da Escassez de memória:* Normalmente a escassez de memória não é dificil de ser identificada, a maioria dos sistemas provê alguma maneira de verificar a utilização de memória de forma precisa. Entretanto, os resultados podem ser enganosos, pois em alguns sistemas, como o Linux, a memória livre pode ser usada para *buffer* de I/O e apesar do sistema apresentar pouca memória livre, não necessáriamente a memória está sendo usada para os processos e para o Sistema Operacional.
- *Quantificar a Demanda de memória para satisfazer as necessidades do sistema:* Uma vez que foi identificada a escassez de memória deve-se mensurar a quantidade de memória a ser alocada para satisfazer as demandas. Os autores do trabalho [Zhao, 2009] trabalham com uma margem de 10% de memória livre e realizam uma análise da taxa de falta de páginas nas MVs para definir a quantidade de memória a ser alocada.
- *Identificação do momento em que a Memória alocada pode ser removida, sem prejudicar o sistema:* Este problema é referente à identificação de picos de utilização. Um mecanismo que implementa a elasticidade de memória deve ser capaz de identificar demandas momentaneas acima da média e adaptar a alocação de memória e remover o excesso de memória após a normalização.

## 4. Estratégia de Alocação de Memória

O mecanismo implementado neste trabalho tem como principal foco solucionar ou amenizar os três problemas abordados na seção anterior. Em estudos anteriores [Baruchi, 2009] foi implementado um protótipo em Perl que utilizou uma técnica diferente para a implementação da elasticidade, baseado na carga de trabalho para alocação ou remoção de memória. A solução proposta tem como base o cálculo da Média Móvel Exponencial (MME) para definir quando a memória deve ser alocada e quando pode ser removida. O uso da MME é bastante comum no mercado financeiro [James, 1968] para identificar a tendência de compra ou venda de ações de uma determinada empresa. Existem, basicamente, duas formas de média móvel:

- Média Móvel Simples (MMS): É a média aritmética das amostras;
- Média Móvel Exponencial: Média ponderada, pois as amostras mais recentes têm maior peso que amostras antigas;

O uso da MME foi escolhido, pois em alguns testes preliminares a MMS não identificou com precisão o término de picos de demanda de memória. Devido a este comportamento ocorreram atrasos para a remoção de memória excedente. O cálculo da MME pode ser observado na equação (1).

$$MME_{atual} = MME_{atual-1} + \alpha * (MEM_{atual} - MME_{atual-1}), \text{ onde:} \quad (1)$$

$$\alpha = 2 / (Qtde\_Amostras + 1);$$

$MEM_{atual}$ = Quantidade de memória usada na ultima medida;

O uso de dois períodos de amostragem (de 5 e 25 segundos) tem como objetivo melhorar a identificação de tendências de alta e baixa demanda de memória e suavizar momentos de picos de utilização. A MME dos últimos 5 segundos tem maior sensibilidade à variação da memória enquanto que a amostra de 25 segundos tem menor sensibilidade. Os pontos de decisão para adicionar ou remover a memória são os pontos de cruzamento entre a MME de 5 segundos ($MME_{(5)}$) e a MME de 25 segundos ($MME_{(25)}$). A identificação da necessidade de adicionar ou remover memória é feita através do sentido em que o cruzamento é realizado. Se a linha da $MME_{(5)}$ cruzar para baixo a linha da $MME_{(25)}$ então a demanda por memória está diminuído, caso ocorra a situação contrária (a linha da $MME_{(5)}$ cruzar para cima a linha da $MME_{(25)}$) então a demanda por memória está aumentando (Figura 2).

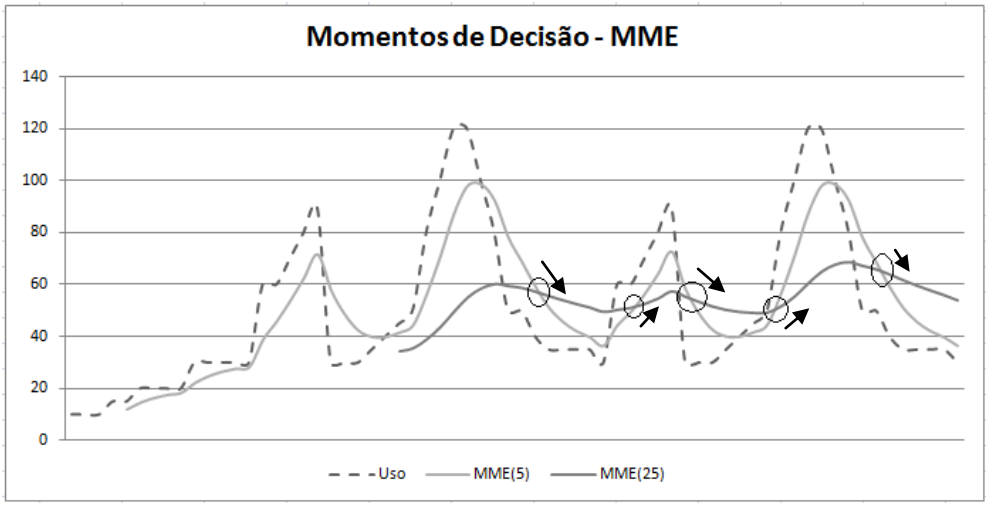



**Figura 2. Ilustração dos pontos de decisão para Adição ou Remoção de memória.**

Na Figura 2, são mostrados alguns pontos de decisão (círculos) e as setas indicam o sentido em que a $MME_{(5)}$ está cruzando a $MME_{(25)}$. A quantidade de memória a ser alocada ou removida nos pontos de decisão é proporcional a quantidade de memória total disponível no sistema. Para que não haja aumento excessivo de falta de páginas, a memória total é acrescida em 10% [Zhao, 2009].

## 5. Resultados Obtidos

Para a verificação da eficiência de alocação da memória utilizando a MME, foi desenvolvido um protótipo em linguagem C que monitora e faz a alocação de memória nas MVs. O protótipo foi dividido em duas partes, uma parte que fica em execução na MV monitorando o uso de recursos e calculando a MME. A outra parte do protótipo localiza-se no MMV e tem a função de realizar as alocações de memória de acordo com as necessidades identificadas. A comunicação entre a MV e o MMV é realizada através de memória compartilhada.

### 5.1. Metodologia

A avaliação do mecanismo foi realizado com dois *benchmarks*. O primeiro *benchmark* utilizado foi o DBench [DBench], um *benchmark* que simula um servidor Samba e de comportamento I/O *Bound*. O DBench permite que a quantidade de usuários

acessando o servidor seja configurado e com isso aumentar ou diminuir a carga de trabalho no ambiente. O segundo *benchmark* utilizado foi a compilação do Kernel do Linux [Kernel]. A compilação do Kernel simula uma aplicação CPU Bound.

Os testes foram repetidos por 15 vezes e como parametro de comparação foram feitos testes com os mesmos *benchmarks* utilizando MVs com memória estática. Após a realização das 15 repetições, foi calculada a média e a distribuição *student*-t [Pearson, 1939] com 95% de confiabilidade. Ao todo foram executados 600 testes (120 testes de compilação do *Kernel* e 480 com o DBench). O DBench foi executado com 50, 75, 100 e 150 clientes simulando acesso ao servidor Samba. Na tabela 1 são apresentadas as configurações das MVs utilizadas para a realização dos testes.

A execução dos testes foi realizada em duas MVs (MV01 e MV02), ambas configuradas de maneira identica (Tabela 1). Os benchmarks foram executados simultaneamente na MV01 e MV02. Para isso, foram criados scripts de automação dos testes, com o objetivo de manter o sincronismo. A execução de forma concorrente é importante, pois raramente um ambiente físico conterá apenas uma única MV.

**Tabela 1. Configuração das MVs utilizadas.**

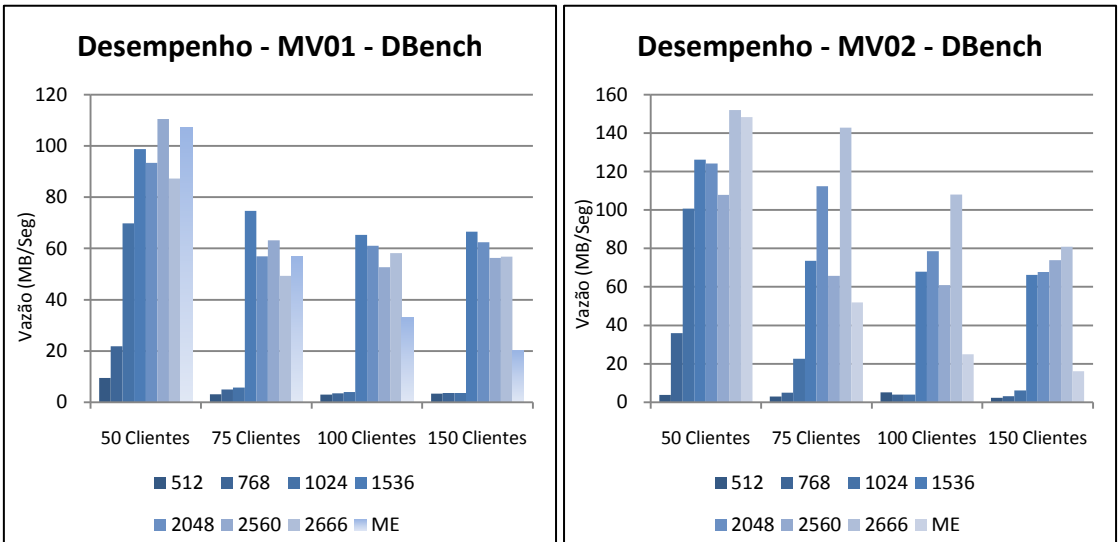
| *Hardware Alocado* | | *Software Instalado* | |
|---|---|---|---|
| Processador (#VCPUs) | 1 | Sistema Operacional | Fedora release 8 |
| Memória (MB) | 512, 768, 1024, 1536, 2048, 2560 e 2666 | Versao do *Kernel* | 2.6.21-2952.fc8xen |
| Disco (GB) | 7,3 | | |
| Rede (# Interfaces) | 1 | | |

## 5.2. Desempenho

A primeira análise a ser realizada é o desempenho das MVs com o mecanismo de elasticidade de memória habilitado (Figura 3). Como pode ser observado o Mecanismo de Elasticidade (ME) com o DBench configurado para 50 e 75 clientes é muito próximo do melhor resultado na MV01 e para 50 clientes na MV02. Nos outros resultados o desempenho fica acima da configuração de 1GB e muito próximo do desempenho da configuração de 1,5GB. Em geral, o desempenho do ME obteve um desempenho mediano para o DBench. A Tabela 2 apresenta os resultados do ME com a distribuição *student*-t.

O desempenho na compilação do Kernel (Figura 4) obteve desempenho próximo da configuração com memória fixa de 1GB na MV01 e desempenho pouco melhor que a configuração fixa de 512MB na MV02. A tabela 3 apresenta os resultados e a distribuição student-t para este *benchmark*.

Em geral, o desempenho com o DBench foi satisfatório. O pior desempenho foi identificado na execução do DBench com 150 Clientes. A maior demanda por memória e processamento na execução do DBench com esta configuração pode ter causado maior demora para a contabilização das necessidades da MV e como consequencia pior desempenho.

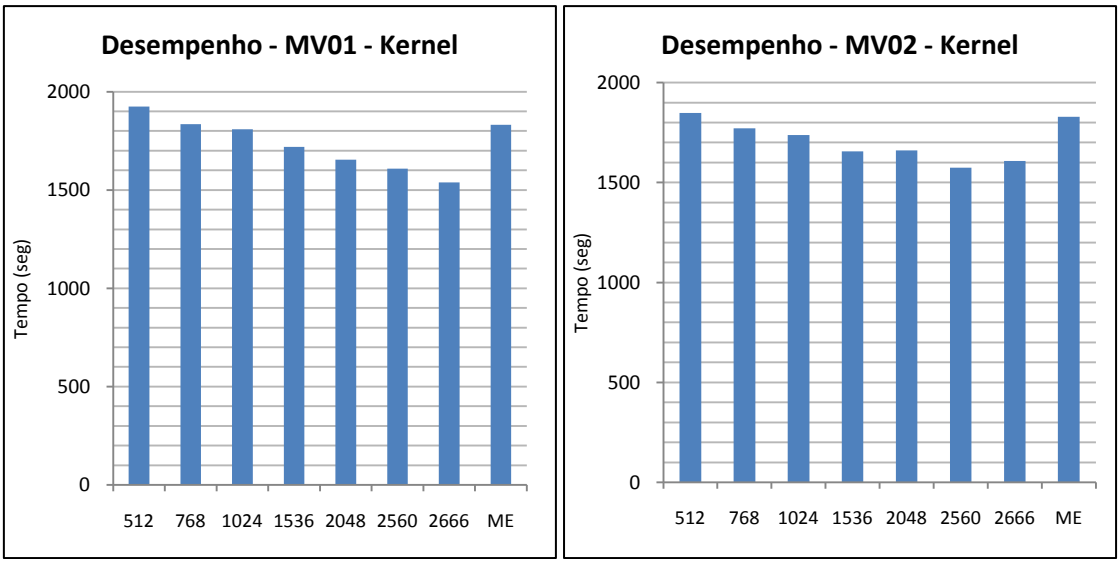



**(a)** **(b)**

**Figura 3. Gráficos do desempenho da MV01(a) e MV02(b) com o DBench.**

**Tabela 2. Resultados do DBench com o ME**

| | MV01 | | MV02 | |
|---|---|---|---|---|
| *Qtde Clientes* | *Média* | *Student-t (95%)* | *Média* | *Student-t (95%)* |
| 50 Clientes | 107,08 | 26,19 | 148,26 | 23,04 |
| 75 Clientes | 56,99 | 15,21 | 51,96 | 20,17 |
| 100 Clientes | 33,09 | 13,64 | 24,96 | 14,96 |
| 150 Clientes | 20,19 | 7,76 | 16,13 | 7,64 |




**(a)** **(b)**

**Figura 4. Gráficos do desempenho da MV01(1) e MV02(b) com a compilação do *Kernel*.**

**Tabela 3. Resultados da Compilação do *Kernel* com a ME**

| MV01 | | MV02 | |
|---|---|---|---|
| *Média (seg)* | *Student-t 95%* | *Média (seg)* | *Student-t 95%* |
| 1831,20 | 14,10 | 1829,53 | 18,23 |

### 5.3. Qualidade de Alocação da Memória

Além de verificar o desempenho do sistema ao utilizar o mecanismo proposto de elasticidade de memória, foi realizado também um estudo sobre a qualidade de alocação de memória. Esta análise consiste em verificar se o mecanismo alocou com precisão a quantidade de memória demandada pela MV. Para isso, foi utilizada uma métrica conhecida como Produto Espaço Tempo (PET) [Denning, 1980].

A qualidade de alocação de memória melhora quando a diferença entre a memória alocada e a memória demandada pela MV é próxima de zero. Quando o valor é proximo de zero o desperdício de memória é menor. A figura 5 ilustra como a qualidade de memória é verificada e a equação (2) mostra como o cálculo da qualidade de alocação de memória é realizado.

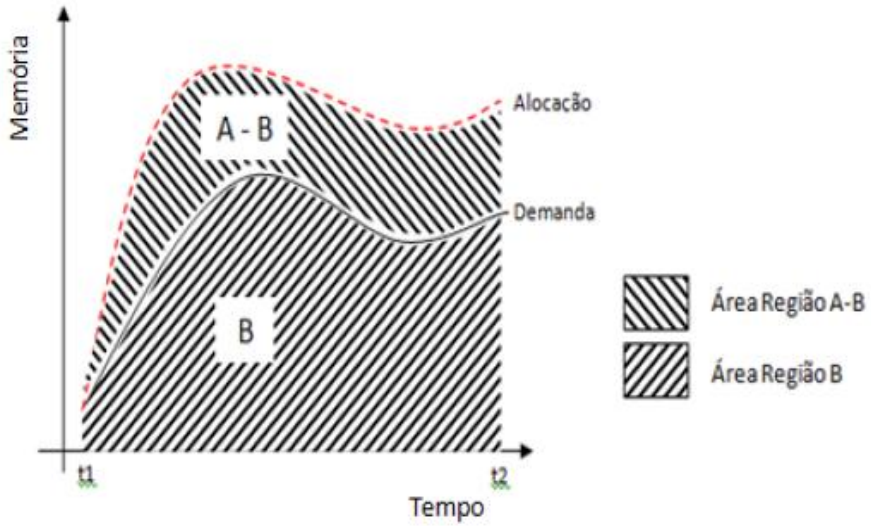



**Figura 5. Qualidade de Alocação de Memória**

$$\text{Área}_{\text{[MB x Segundo]}} = \int_{t1}^{t2} y(t)dt \qquad (2)$$

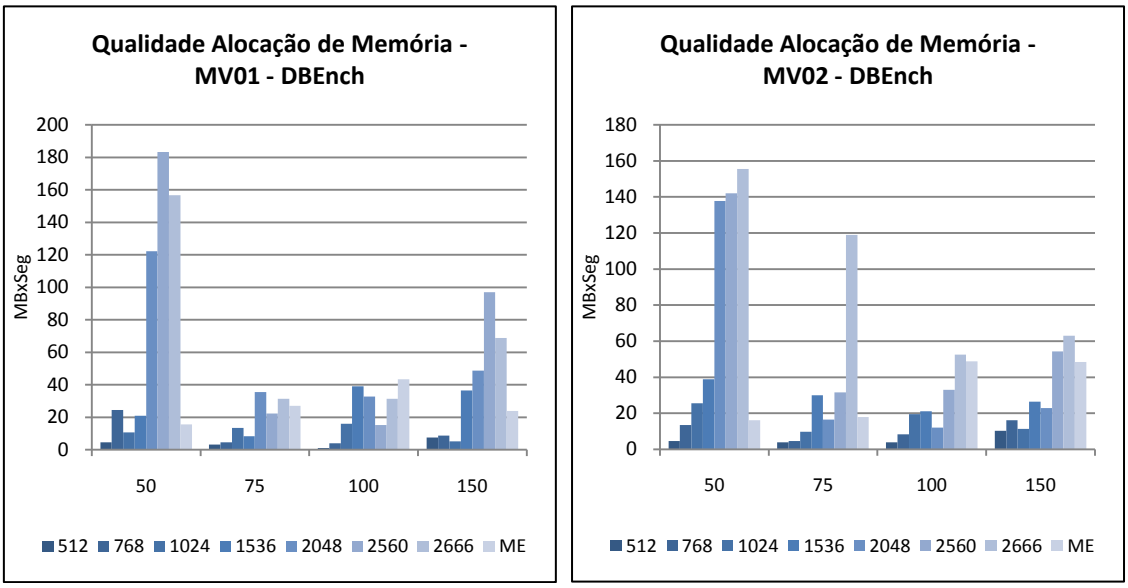
Onde:

t1 e t2: Tempo total de execução do benchmark;

(y)t: Curva que representa a alocação ou a demanda da memória;

A região representada na figura 5 como A-B é a região que representa a qualidade de alocação de memória. A região B do gráfico representa a quantidade de memória demanda pela MV (memória utilizada). Nota-se que, com uma grande

quantidade de memória alocada, mas com baixa demanda, a diferença entre as duas áreas (região A-B) tende a ser maior e uma diferença igual a zero indica que o ambiente pode estar enfrentando uma situação de *trashing*. É importante observar que o ideal é que sempre haja uma quantidade de memória livre (conforme comentado anteriormente, algo em torno de 10%). A unidade de medida do cálculo da qualidade de alocação é em MBxSegundo (Megabytes Segundo).




**(a)** **(b)**

**Figura 6. Qualidade de alocação de memória na MV01 (a) e MV02 (b).**

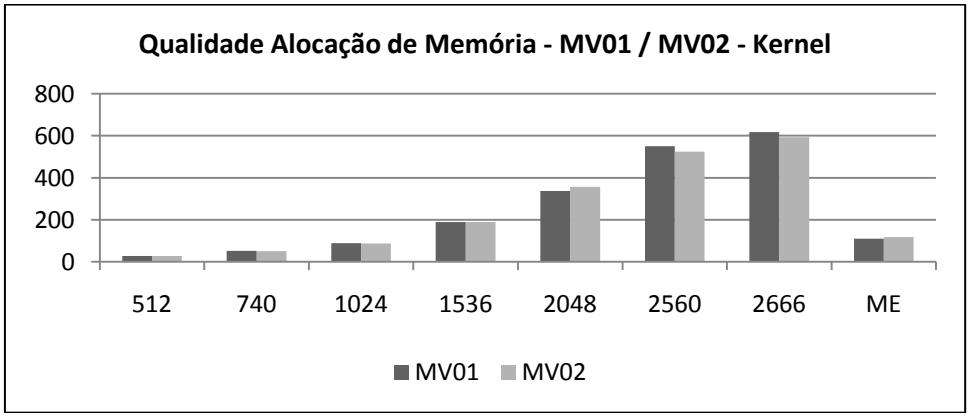



**Figura 7. Qualidade de alocação de memória na MV01 e MV02**

As figuras 6a e 6b apresentam a qualidade de alocação da memória para o *benchmark* DBench. A alocação de memória para 50 e 75 clientes foi muito próxima da memória demandada pela MV. Para as outras configurações de clientes, 75 e 100, a alocação de memória ficou próxima das alocações fixas de 2,6GB. Nestas cofigurações o desperdício de memória foi maior quando comparado com quantidades menores de clientes.

Na compilação do Kernel (figura 7), nota-se que houve menor desperdício de memória durante as execuções do benchmark. A alocação de memória foi 20% superior quando comparado a alocação de 1GB na MV01 e 25% superior na MV02 quando comparado com a mesma configuração de 1GB.

## 6.Conclusões e Trabalhos Futuros

Este trabalho apresentou um mecanismo para a implementação da Elasticidade de Memória em MVs. A Elasticidade de recursos é uma caracteristica muito atraente em ambientes virtualizados e essencial em ambientes que implementam a computação em nuvem. Foram apresentados os principais problemas da implementação da elasticidade de memória e foi proposta uma solução utilizando a Média Móvel Exponencial (MME). O uso da MME é comum no mercado financeiro para a análise de tendências em ações de empresas. Para a demonstração da qualidade de alocação de memória foi utilizado o Produto Espaço Tempo (PET), cujo objetivo é mensurar a quantidade de memória demandada e a alocada no ambiente.

Os testes realizados com o mecanismo mostraram que o uso da MME pode ser bastante útil em cargas de trabalhos não muito extremas e em aplicações I/O Bound como o DBench. A compilação do Kernel não obteve um bom desempenho nos testes realizados. A principal causa do pior desempenho na compilação do Kernel se deve principalmente ao *Balloon Driver*. Constantes ativações no *Balloon Driver* podem causar danos ao desempenho, pois o *Balloon* ativa os algoritmos de paginação do Sistema Operacional. A ativação desses algoritmos com muita frequência é ruim, pois possuem mais prioridade (são executados em espaço de *Kernel*) que os outros processos no sistema. Os danos causados no DBench foram menores, pois o DBench não possui tanta dependência de CPU como a compilação do *Kernel*.

Alguns trabalhos para melhorar a elasticidade de memória estão sendo analisados, como o uso de Inteligencia Artificial e a predição do uso de memória. Apesar de estratégias como o uso da MME serem interessantes, a implementação de um mecanismo de predição pode ser mais eficiente pois evita que as faltas de páginas ocorram. Outro ponto a ser explorado é um mecanismo que altere a quantidade de memória da MV sem causar maiores danos ao desempenho do sistema.

## 7. Referências


Arcangeli, Andrea; Eidus, Izik; Chris Wright (2009). "Increasing Memory Density by Using KVM". Proceedings of the 2009 Linux Symposium.

Armbrust, Michael. et. al. (2009). "Above the Clouds: A Berkeley View of Cloud Computing". EECS Department, Technical Report No. UCB/EECS-2009-28.

Barham, P., B. Dragovic, et al. (2003). "Xen and the Art of Virtualization". 19th ACM Symposium on Operationg System Principles. Bolton Landing, NY, USA: ACM Press. p. 164-177.

Baruchi, Artur; Midorikawa, Edson (2009). “Gerência de Memória Adaptativa no XEN”, Trabalhos em Andamento, VI Workshop de Sistemas Operacionais, Bento Gonçalves, RS.

Bertogna, Leandro Mario. et. al. (2009)."Dynamic on Demand Virtual Clusters in Grid". Euro-Par 2009 Workshop - VHPC'09. p. 13-22.

Buyya, R. et al. (2008) "Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities". In Proc. 10th IEEE International Conference on High Performance Computing and Communications (HPCC 2008), IEEE CS Press, Sept. 25–27.

DBench. http://dbench.samba.org/

Denning, P. J. (1980) "Working sets Past and Present". IEEE Transactions on Software Engineering. p. 64-84.

Goldberg, Robert P. (1974). "Survey of Virtual Machine Research". IEEE Computer. pp. 34-45.

James, F. E. (1968). "Monthly Moving Averages--An Effective Investment Tool?". Journal of Finantial and Quantitative Analysis, Cambridge University. p. 315-326.

Kandururi, Balachandra; Paturi, Ramakrishna; Rakshit, Atanu. "Cloud Security Issues". IEEE International Conference on Services Computing, 2009.

Kernel. http://kernel.org/

Kivity, Avi. et. al.(2007). "KVM: The Linux Virtual Machine Monitor", Proceedings of The Linux Symposium, Ottawa, Ontario.

Mirtchovski, Andrey; Ionkov, Latchesar. (2007). "KvmFS: Virtual Machine Partitioning For Cluster and Grids". In Ottawa Linux Symposium Proceedings.

Moskovisky, Alexander; Pervin, Artem; Walker, Bruce. (2008). "Dynamic Resources Management of Virtual Appliances on a Computational cluster". Euro-Par 2008 Workshop - VHPC'08. p. 33-42.

Muntean, T. (1994). "A generic multi virtual machines architecture for distributes parallel operating system design". IEEE Heterogeneous Computing Workshop. Abril.

Parkhill, Douglas F. (1966). "The challenge of computer utility". Addison-Wesley Educational Publisher.

Pearson, E. S. (1939) "Student as a Statistian". Biometrika, v. 20. p. 210-250.

Rosenblum, M.; Garfinkel, T. (2005). "Virtual Machine Monitors: Current Technology and Future Trends. Computer". p. 39-47, Maio.

Vecchiola, Christian; Pandey, Suraj; Buyya, Rajkumar. (2009). "High-Performance Cloud Computing: A view of Scientific Applications". Proceedings of the 10th International Symposium on Pervasive Systems, Algorithms and Networks.

Waldspurger, C. A. (2002). "Memory Resource Management in VMWare ESX Server". Proceeding of the 5th Symposium on Operating System Desing and Implementation. Boston, MA. p. 181-194.

Zhao, Weiming; Wang, Zhenlin (2009). "Dynamic Memory Balancing for Virtual Machines". ACM/Useninx International Conference on Virtual Execution Environments. Washington, DC. p. 21-30.