UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

ESTUDO COMPARATIVO DE DESEMPENHO EM AMBIENTE TRADICIONAL E VIRTUALIZADO APLICADO A BANCO DE DADOS EM PLATAFORMA *X86*

ADRIANA SILVA NEIVA

ORIENTADOR: GEORGES DANIEL AMVAME NZE

DISSERTAÇÃO DE MESTRADO EM ENGENHARIA ELÉTRICA

PUBLICAÇÃO: 069/10

BRASÍLIA/DF: Agosto 2010

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

ESTUDO COMPARATIVO DE DESEMPENHO EM AMBIENTE TRADICIONAL E VIRTUALIZADO APLICADO A BANCO DE DADOS EM PLATAFORMA *X86*

ADRIANA SILVA NEIVA

DISSERTAÇÃO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE.

APROVADA POR:

_____

Prof. Georges Daniel Amvame Nze, Dr. (FGA/UNB)
(Orientador)

_____

Prof. Flávio Elias Gomes de Deus, Dr. (ENE/UNB)
(Examinador Interno)

_____

Prof. Fernanda Lima, Dra. (CIC/UNB)
(Examinador Externo)

BRASÍLIA/DF, 30 DE AGOSTO DE 2010

FICHA CATALOGRÁFICA

NEIVA, ADRIANA SILVA

Estudo comparativo de desempenho em ambiente tradicional e virtualizado aplicado a Banco de Dados em Plataforma *x86*.[Distrito Federal] 2010.

xvii, 101p., 210 x 297 mm (ENE/FT/UnB, Mestre, Dissertação de Mestrado – Universidade de Brasília. Faculdade de Tecnologia.

Departamento de Engenharia Elétrica

1.Virtualização 2. *x86*

3.Banco de Dados 4.Monitor de Máquina Virtual

I. ENE/FT/UnB II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

NEIVA, A. S. (2010). Estudo comparativo de desempenho em ambiente tradicional e virtualizado aplicado a Banco de Dados em Plataforma *x86*. Dissertação de Mestrado em Engenharia Elétrica, Publicação 069/10, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 101p.

CESSÃO DE DIREITOS

AUTOR: Adriana Silva Neiva

TÍTULO: Estudo comparativo de desempenho em ambiente tradicional e virtualizado aplicado a Banco de Dados em Plataforma *x86*.

GRAU: Mestre ANO: 2010

_____

Adriana Silva Neiva
SCN Quadra 2 Bloco F Térreo.
71.712-906 Brasília – DF – Brasil.

AGRADECIMENTOS

A Deus pela sua presença incondicional no meu dia a dia e por tudo que pude realizar.

Ao Prof. Dr. Georges Daniel Amvame Nze pela sua disponibilidade, generosidade e valiosa orientação.

Ao Mestre Sinésio Teles de Lima pelo apoio, participação e incentivo durante todo o trabalho.

Aos amigos Carlos Mario Vieira de Melo Filho, Evandro Jurente de Sousa Gil, Alan William da Silva, Renato Costa Pereira, Luciano Ricardi Scorsin, Viviane Rosa de Oliveira, Rodrigo Pinheiro do Santos, Dayler Losi de Morais e Gilberto Ribeiro Soares Junior pela parceria e companheirismo durante a montagem do laboratório e execuções dos testes práticos e comparativos.

Dedico este trabalho a **Deus** e a minha mãe **Maria de Lourdes Botelho Neiva**, pelo amor incondicional, força e incentivos constantes, e a todos meus familiares e amigos, fontes de inspiração e energia.

RESUMO

ESTUDO COMPARATIVO DE DESEMPENHO EM AMBIENTE TRADICIONAL E VIRTUALIZADO APLICADO A BANCO DE DADOS EM PLATAFORMA *X86*

Autor: Adriana Silva Neiva

Orientador: Georges Daniel Amvame Nze

Programa de Pós-graduação em Engenharia Elétrica

Brasília, agosto de 2010

Esta dissertação de mestrado apresenta um estudo comparativo de desempenho em ambiente tradicional e virtualizado aplicado a banco de dados em plataforma *x86*. O objetivo deste trabalho é identificar a sobrecarga de uso de recursos computacionais em banco de dados decorrente da virtualização. Por meio de estudo comparativo foram realizados testes e análises para identificar o real impacto da camada de virtualização em relação à capacidade de tratar transações em banco de dados.

ABSTRACT

COMPARATIVE STUDY OF PERFORMANCE IN TRADITIONAL AND VIRTUALIZED ENVIRONMENTS APPLIED TO DATABASE IN X86 PLATAFORM

Author: Adriana Silva Neiva

Supervisor: Georges Daniel Amvame Nze

Programa de Pós-graduação em Engenharia Elétrica

Brasília, august of 2010

This dissertation presents a comparative study of performance in traditional and virtualized servers applied to the database on x86 platform. The objective is to identify the overhead of using computing resources in the database due to virtualization. Through comparative study tests and analysis were conducted to identify the real impact of virtualization layer over the ability to handle transactions in the database.

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE SÍMBOLOS, NOMENCLATURA E ABREVIAÇÕES

AMD - *Advanced Micro Devices*

Bit - *Binary Digit*

DSS - *Decision Support System*

E/S - Entrada/Saída

Gb/s - *Gigabits* por segundo

JDBC - *Java Database Conectivity*

Mb/s - *Megabits* por segundo

MB/s - *Megabytes* por segundo

MMV - Monitor de Máquina Virtual

MV - Máquina Virtual

OLAP - *On-Line Analytical Processing*

OLTP - *On-Line Transation Processing*

RAM - *Random Access Memory*

RPM - Rotações por minutos

SAS - *Serial Attached SCSI*

SGBD - Sistema de Gestão de Base de Dados

SO - Sistema Operacional

SOE - *Swingbench Order Entry*

TI - Tecnologia da Informação

TPC-C - *Transaction Processing Performance Council*

UCP - Unidade Central de Processamento

USB - *Universal Serial Bus*

1 - INTRODUÇÃO

Atualmente as empresas estão em um palco de disputas de margens de rentabilidade sob os olhares atentos de um público que cobra, cada vez mais, simplicidade e qualidade dos produtos ofertados a um custo cada vez menor. Como efeito, todas as áreas de uma empresa são forçadas a fazer mais com menos, ou ainda, a oferecer mais produtos e serviços mantendo-se as restrições orçamentárias (DESAI, 2007).

Outro ponto também relevante é a própria comoditização[1] dos produtos. Esta questão tem levado as empresas a procurar oportunidades pelo diferente, a se preocupar mais em perceber o mercado, o que o cliente realmente deseja, e a ter agilidade para responder a esses desejos de forma a contribuir para o posicicionamento diferenciado da empresa no mercado.

Uma das soluções tecnológicas que se propõe a ajudar a área de Tecnologia da Informação (TI) a vencer os desafios de custo e agilidade é a virtualização de servidores. A virtualização de servidores é uma técnica que permite que uma única máquina física se apresente à rede e aos usuários como várias máquinas independentes, também chamadas de máquinas virtuais (MV). A idéia geral é permitir que muitos sistemas operacionais (SO) independentes possam ser executados simultaneamente na mesma máquina física. O princípio básico é o melhor aproveitamento de recursos: ao invés de *n* servidores com percentual de utilização de *x* é possível ter um único servidor com um percentual de uso de aproximadamente *n.x* (CARISSIMI, 2008).

Uma primeira vantagem da virtualização é o fato de que o Centro de Dados[2] poderá ter menos servidores do que possui hoje, diminuindo a complexidade da gestão, consumo de energia, necessidade de espaço, entre outros, o que contribui para a redução de custos.

Outra vantagem é o compartilhamento dos recursos de processamento, memória e disco da máquina física para as MV. Recursos ociosos em uma máquina física poderão ser liberados para as MV que necessitarem de mais recursos, ou ainda poderão ser utilizados para atender novas necessidades. Essa alocação tipicamente ocorre em minutos ou horas o

[1] Por comoditização entende-se a dificuldade cada vez maior que um produto tem para se diferenciar de outro, tanto do ponto de vista técnico quanto do ponto de vista de utilidade.
[2] Centro de Dados, em inglês *Data Center*, é o centro de processamento de dados até recentemente chamado de CPD (Centro de Processamento de Dados).

que permite reduzir substancialmente o tempo de liberação do recurso, se comparado com o modelo tradicional (servidor não virtualizado) que demanda dias por envolver, por exemplo, tempo para contratação e tempo da instalação física.

Apesar das vantagens relacionadas, e de as tendências de mercado sugerirem que a virtualização já não é mais uma opção que as empresas podem ou não escolher, mas sim para onde todas devem convergir, sabe-se que a virtualização não é ilimitada e benéfica para qualquer escala (GARTNER, 2007). Hoje se observa alguma restrição dessa tecnologia em relação às aplicações que consomem todos, ou quase todos, os recursos de um servidor, ou que tenham necessidade de leitura e escrita intensas, ou ainda que exijam acesso a recursos de aceleração gráfica em três dimensões (BITTMAN, et al., 2008).

Entre as camadas típicas de uma aplicação (apresentação, negócios e banco de dados), as grandes empresas de telecomunicações vêm utilizando a virtualização em plataforma *x86*[3] para as camadas de apresentação e negócios. Devido às altas taxas de leitura e escrita comumente exigidas para a camada de banco de dados, essas empresas têm sido mais conservadoras e praticado o uso de servidores dedicados (tradicionais) como servidores de banco de dados. Uma dessas empresas, por exemplo, de um total de três mil quatrocentos e quarenta servidores *x86* (máquinas físicas), cento e vinte são dedicados à virtualização, suportando uma mil duzentas e quarenta e sete MV, todas para atendimento exclusivo às camadas de apresentação e negócios (SILVA, 2010).

Por outro lado, alguns *benchmarks* e artigos técnicos têm sido publicados com o intuito de mostrar que os *softwares* de virtualização e as tecnologias *x86* têm sofrido evoluções focadas em atingir desempenho e capacidade de tratamento de transações de bancos de dados próximos ao de um servidor tradicional.

O fornecedor *Oracle*[4], por exemplo, em parceria com o Grupo *Tolly*, constatou que sua solução de virtualização, o *Oracle VM*[5], tem uma eficiência em torno de 92,5% a 93,6% se comparada com o ambiente tradicional. Os testes realizados aplicaram uma carga

[3] Em informática, *x86* é o nome genérico dado à arquitetura de processadores baseados no *Intel 8066* da *Intel Corporation.*
[4] A *Oracle* é uma companhia que desenvolve *softwares* corporativos. O seu principal produto é o Sistema de Gestão de Base de Dados relacional chamado *Oracle*.
[5] *Oracle VM* é a solução de virtualização da *Oracle* que usa a técnica de paravirtualização e suporta aplicações de 32 e 64 *bits.*

em um banco de dados *Oracle* através da ferramenta *Swingbench*[6] considerando um total de 30 e 50 usuários concorrentes acessando o banco de dados (THE TOLLY GROUP, 2008) (PINTO, 2009).

O fornecedor *VMware*[7], em documento técnico que explora a potencialidade das tecnologias *x86* para apoiar a virtualização, aponta uma oportunidade de ganho de eficiência de 3% na capacidade de tratamento de transações de bancos de dados *Oracle*, na versão *11G R1*, quando comparando a virtualização de memória assistida por *hardware* à virtualização de memória somente por *software* (VMWARE, 2008-2009).

Em *benchmark* comparativo das versões *VMware ESX 3.4*[8]e *VMware ESX 4.0*[9], também realizado pela *VMware*, foi apurado um ganho de eficiência de 24% para o *VMware ESX 4.0* em testes realizados com servidores de dois processadores atendendo à camada de banco de dados. Os mesmos testes aplicados à servidores com oito processadores, atingiram eficiência de 28% para o *VMware ESX 4.0* (VMWARE, 2009).

O fornecedor *Intel*[10] realizou testes de laboratórios de seu processador *Intel Xeon 7500*[11], onde o ambiente tradicional atingiu uma taxa média de transações por segundo de 3.730,07 e o ambiente virtualizado de 3.538,64, o que conferiu ao ambiente virtualizado uma eficiência de 94,86%. O Sistema de Gestão de Base de Dados (SGBD) utilizado foi o *SQL Server*[12] e a solução de *software* de virtualização o *VMware ESX 4.1* da *VMware* (PROWESS, 2010).

No entanto, não foram encontradas na literatura referências acerca do tempo de resposta obtido isoladamente pelas transações que compuseram os testes, e nem demonstrativos dos consumos computacionais que a camada de virtualização impôs aos principais recursos virtualizáveis de um servidor: Unidade Central de Processamento (UCP), memória e armazenamento.

---

[6] O *Swingbench* é uma ferramenta desenvolvida por *Dominic Giles*, do Grupo de Soluções de Banco de Dados da *Oracle*, com o propósito de provocar cargas em bancos de dados *Oracle* e exercitar o consumo de recursos físicos do servidor.
[7] A *VMware* é uma empresa de propriedade da *EMC Corporation* que fornece soluções de virtualização via *software*.
[8] O *VMware ESX 3.4* é o *software* de virtualização da *VMware* para plataforma *x86*, antecessor à versão 4.0.
[9] O *VMware ESX 4.0* é o *software* de virtualização da *VMware* para plataforma *x86*, sucessor da versão 3.4.
[10] A *Intel* é uma empresa de tecnologia no segmento de semicondutores de *chips*, placas mãe, controladores de interfaces de rede, circuitos integrados e memórias.
[11] O *Intel Xeon 7500* é um processador da família *Nehalem* da *Intel*.
[12] O *SQL Server* é um Sistema de Gestão de Base de Dados de propriedade da *Microsoft* e que pode ser instalado somente em plataforma da família *Windows Server*.

Esta dissertação pretende explorar esses dois aspectos - tempo de resposta e consumo de recursos computacionais - em ambiente virtualizado aplicado à camada de banco de dados em plataforma *x86*. Através da realização de testes, serão comparados os resultados obtidos no ambiente tradicional e no virtualizado com o intuito de identificar a real sobrecarga de processamento na camada de banco de dados decorrente da virtualização.

Para tanto, o trabalho está estruturado da seguinte forma: a seguir são apresentados os principais fatores motivadores da virtualização e deste trabalho, e os objetivos específicos deste estudo com as respectivas métricas que serão utilizadas para fazer o comparativo. No Capítulo 2, são apresentadas as características e evoluções das soluções de virtualização. No Capítulo 3, apresenta-se o cenário da carga escolhida para os testes, a relação dos *hardwares* e *softwares* utilizados e o detalhamento da metodologia aplicada. No Capítulo 4, são apresentados os resultados obtidos nos testes realizados. No capítulo 5 são apresentadas as conclusões alcançadas a partir dos testes realizados e as sugestões e recomendações para futuros trabalhos a serem desenvolvidos.

1.1 - MOTIVAÇÃO E RELEVÂNCIA DO ESTUDO

Uma pesquisa realizada pela *Network World* (GAREISS, 2009), identificou que a virtualização de servidores é motivada, em ordem de importância, pelas seguintes razões: aumentar a taxa de utilização de servidores, reduzir os custos dos Centros de Dados, aperfeiçoar os procedimentos de recuperação de desastres e de *backup*, criar ambientes mais flexíveis para desenvolvimento e testes de *software*, e reduzir custos de administração de TI. As duas primeiras razões têm sido as mais praticadas (BITTMAN, et al., 2008), por estarem aderentes à atual situação ambiental e econômica mundial, onde as empresas têm buscado a redução dos consumos de energia, refrigeração e espaço (tão escassos ultimamente) e a adequação das necessidades de infraestrutura às limitações orçamentárias e à diretriz (ainda que antagônica) de fazer mais com menos.

Mas independente do fator motivador, o fato é que a virtualização de servidores implica em múltiplas máquinas virtuais (MV) compartilhando acesso a um conjunto único de recursos de *hardware*. Esse compartilhamento, se não bem planejado ou estruturado, pode resultar em gargalos de desempenho inaceitáveis, causando uma incapacidade de

cumprir o objetivo básico e mandatório de desempenho. Quando isso ocorre, a carga de trabalho ou a aplicação em questão não deveria ser virtualizada (GAMMAGE, et al., 2008), independente dos ganhos que a virtualização traria para a redução de ocupação de espaço, diminuição de custos, maior agilidade, dentre outros.

Diversas aplicações utilizadas em empresas de telecomunicações, como Sistemas de Autenticação de Provedores da Internet, Sistemas de Mediação de Chamadas e Sistemas de Faturamento, entre outros, dependem do desempenho da camada de banco de dados. Sob a ótica da capacidade das soluções de virtualização suportarem transações de bancos de dados, conforme exposto, há *benchmarks* e estudos que apontam que essa capacidade está próxima da suportada por um servidor tradicional. Todavia, não foram encontradas na literatura referências acerca de dois importantes aspectos para a tomada de decisão quanto a adotar a virtualização na camada de banco de dados: o tempo de resposta das transações do banco de dados e o consumo de recursos computacionais decorrentes da virtualização.

O tempo de resposta tem impacto direto na percepção do usuário quanto aos serviços prestados por TI, e sua degradação compromete o acordo de nível de serviço com enfoque em desempenho, e põe em risco os negócios da organização que dependem da aplicação impactada. Desta forma, ao optar pelo uso da virtualização em camada de banco de dados, a TI deve assegurar a manutenabilidade do tempo de resposta dentro do acordado com a organização, a fim de preservar a estabilidade da aplicação e dos negócios.

O consumo adicional de recursos do servidor, se chegar a patamares que provoquem gargalos ou contenções, sinaliza riscos potenciais ou eminentes para as aplicações que compartilham o mesmo servidor físico em um ambiente virtualizado, visto que todas serão impactadas por esses gargalos. Esse impacto pode ser em maior escala se o perfil da aplicação que compartilha o servidor físico for similar ao aplicativo ofensor, isso porque eles competem pelo mesmo recurso, ou em menor escala quando o perfil for complementar, pois o uso do recurso ocorrerá de forma intercalada.

Diante do exposto, a opção pela instalação de um SGBD em ambiente virtualizado deve levar em consideração a sobrecarga que a camada de virtualização acrescenta, de forma a apurar se seu impacto no desempenho da aplicação e da virtualização permite (ou não) a sua adoção. Esta dissertação propõe explorar esse aspecto de forma a prover, ao profissional de TI, material técnico de referência por meio do qual se tenha visibilidade aos impactos no tempo de resposta das transações do banco de dados e no consumo de recursos

computacionais decorrentes da virtualização. Espera-se, com este estudo, que os resultados e conclusões obtidos contribuam, de alguma forma, para o planejamento e a estruturação da adoção da virtualização para banco de dados pelos profissionais de TI das grandes empresas de telecomunicações.

1.2 - OBJETIVOS

O objetivo deste trabalho é realizar um estudo comparativo de desempenho em ambiente tradicional e virtualizado aplicado a banco de dados em plataforma *x86,* de forma a identificar a real sobrecarga de processamento decorrente da virtualização. As técnicas de virtualização a serem abordadas são as três atualmente disponíveis para a plataforma *x86*: virtualização completa sem assistência por *hardware*, paravirtualização e a virtualização completa assistida por *hardware*. Os próximos capítulos oferecem maiores detalhes acerca dessas três técnicas.

As métricas a serem utilizadas para aferir a sobrecarga no ambiente virtualizado são: (i) capacidade de tratar transações de bancos de dados; (ii) tempo de resposta da transação; (iii) consumo de UCP; (iv) uso de memória; (v) taxas de leitura e gravação em disco; (vi) capacidade de atendimento a processos e segmentos[13]; (vii) tratamento de interrupções; e (viii) tratamento de trocas de contextos.

Pretende-se com o resultado deste trabalho prover material técnico de referência por meio do qual a sobrecarga inerente à virtualização aplicada à camada de banco de dados possa ser entendida e, por conseqüência, ponderada quando da avaliação de sua adoção. Resultados que apontem uma sobrecarga em patamares similares ao ambiente tradicional, ou seja, que não comprometam o objetivo básico e mandatório de desempenho, podem favorecer uma tomada de decisão por parte das grandes empresas de telecomunicações em estender a adoção da virtualização para a camada de banco de dados.

[13] Segmentos, em inglês *threads,* são o mesmo que linhas de execução. É uma forma de um processo dividir a si mesmo em duas ou mais tarefas que podem ser executadas simultaneamente. O suporte a segmentos é fornecido pelo sistema operacional no nível de *kernel*.

2 - REFERENCIAL TEÓRICO

De forma geral, um sistema de virtualização é um *framework* - uma metodologia - para divisão de um recurso computacional em múltiplos ambientes, aplicando-se um ou mais conceitos ou tecnologias tais como particionamento de *hardware*, compartilhamento de recursos, simulação de máquinas completas ou parciais, emulação e outros. Aplica-se a uma gama de recursos de infraestrutura de TI, tais como subsistemas de armazenamento (*storages* e fitotecas robotizadas), elementos de rede (roteadores, *switches* e *firewalls*) e servidores.

Focando sua aplicabilidade a servidores, a virtualização consiste em permitir que múltiplos SO (como por exemplo, *Linux*[14], *Windows 2003*[15], *Windows 2008*[16]) possam ser executados simultaneamente em um mesmo servidor físico. Cada um desses SO é executado em um espaço isolado conhecido como máquina virtual (MV), independente das demais (CHEN, 2008). Com o isolamento da MV, cada uma não impacta a outra, mesmo se seu SO for desligado, reiniciado ou sofrer uma pane. O isolamento também protege contra acesso não autorizado. Os níveis apropriados de isolamento da MV ajudam a definir o nível de conforto e a possibilidade de executar determinadas cargas de trabalho no mesmo servidor físico como, por exemplo, a produção e teste, duas diferentes organizações, dois diferentes departamentos (PHELPS, et al., 2007).

2.1 - EVOLUÇÃO HISTÓRICA DA VIRTUALIZAÇÃO

Embora bastante difundido e discutido nos últimos nove anos, a virtualização teve sua origem em 1960 com a IBM. Detentora de equipamentos *mainframe* com grande poder de processamento, a IBM percebeu que era preciso prover maior robustez aos *mainframes*, pois os mesmos tinham condições de suportar uma maior carga de processamento do que normalmente lhes era imposta. Para tanto, a IBM passou a desenvolver técnicas de uso compartilhado dos recursos do *mainframe*, e em 1960 foi liberado o primeiro *mainframe*

[14] *Linux* é o termo geralmente usado para designar qualquer sistema operacional que utilize o núcleo *linux*. Seu código fonte fica disponível para que qualquer pessoa possa utilizá-lo, estudá-lo, modificá-lo ou distribuí-lo, respeitando-se o código de licenciamento *GPL (General Public License)*.
[15] O *Windows 2003* é um sistema operacional desenvolvido pela *Microsoft* para a plataforma *x86*, cujo lançamento ocorreu em abril de 2003.
[16] O *Windows 2008* é um sistema operacional desenvolvid pela Microsoft para a plataforma *x86*, cujo lançamento ocorreu em fevereiro de 2008.

com capacidade para tratar dois ou mais usuários de forma simultânea (OGDEN, 2006) (MENASCÉ, 2005).

Mais tarde, em 1966, a IBM lança o CP-40 (IBM System S/360-40) e imediatamente após seu lançamento o CP-40 é melhorado dando origem ao CP-67 (IBM System S/360-67). São incorporadas na arquitetura do CP-67 técnicas para tratamento de uma virtualização completa, no qual é permitida a execução simultânea de vários sistemas operacionais, ocupando cada um uma unidade lógica chamada de máquina virtual (MV) ou sistema convidado ou sistema hóspede. O SO utilizado no CP-67 foi o CP/CMS que foi distribuído como código aberto e sem suporte por parte da IBM (OGDEN, 2006).

A partir de 1970 a IBM lança a série 370, curiosamente sem as técnicas de virtualização. Percebendo o impacto negativo que a ausência da virtualização causava às expectativas e necessidades dos clientes, a IBM reverteu a situação em 1972, incorporando a virtualização na série 370 e substituindo o SO CP/CMS pelo Virtual Machine (VM)/370. Diferentemente do CP/CMS, o VM/370 é liberado no mercado com o suporte oficial pela IBM (OGDEN, 2006) (MENASCÉ, 2005).

Os primeiros VM/370 lançados apresentaram problemas relativos a desempenho e algumas incompatibilidades nas migrações das aplicações de um VM/360 para o VM/370 devido essa nova arquitetura não suportar algumas facilidades do VM/360. Os problemas de desempenho foram decorrentes da técnica de virtualização adotada para o VM/370 onde entre o *hardware* e as MV havia um *software* monitor de máquinas virtuais (MMV) responsável por realizar todas as instruções privilegiadas, tais como operações de entrada/saída (E/S), acesso a memória, dentre outros. Desta forma, ao se migrar um aplicativo que antes executava num único *mainframe* para uma MV, nitidamente se notava o problema de desempenho em virtude das instruções privilegiadas agora requererem um nível a mais, o MMV, para serem efetivadas (OGDEN, 2006) (MENASCÉ, 2005).

Como estratégia para resolver os problemas e as incompatibilidades relatados, a IBM lança o VM/370-XA, agora com compatibilidade total para máquinas virtuais VM/360 e VM/370. Nesta versão as MV passam a executar quase todas as instruções privilegiadas (exceto as operações de E/S) minimizando, assim, os problemas de desempenho. Posteriormente, a IBM lança o Virtual Machine/Enterprise Systems Architecture (VM/ESA) agora melhorando substancialmente a questão desempenho. Desta

data em diante, a IBM passa a evoluir seus sistemas operacionais sem grandes inovações (OGDEN, 2006).

Entre 1972 e 1980 o mercado permanece praticamente estável. Nos anos 80 a virtualização perde espaço para os computadores baseados em micro processadores em virtude de seu baixo preço e do advento da arquitetura cliente/servidor. Essa situação perdura até os anos 90, quando a evolução da tecnologia dos servidores de menor porte (*x86*) atinge um nível de capacidade de realizar cálculos aritméticos e lógicos e tratar maiores volumes de dados o suficiente para atender sistemas de missão crítica ou consolidações de servidores. Neste momento, o assunto virtualização novamente ganha força. Os fabricantes passam a investir em soluções para a arquitetura *x86*, sistemas operacionais *Linux e Windows*[17].

A IBM, para acompanhar esta nova tendência, lança em 2000 o zVM onde o *mainframe* passa a suportar o *Linux*. Devido ao alto custo de aquisição e manutenção do *mainframe*, o zVM não atrai muito o mercado (OGDEN, 2006).

Para a plataforma *x86*, em 1999 a *VMware* anuncia o desenvolvimento do *VMware Virtual Plataform* para *intel* 32 bits e em 2001 lança o primeiro produto para servidor. O modelo adotado é a virtualização completa com uso de *software* MMV. Nesse mesmo modelo são lançados, posteriormente, produtos da *Parallels*[18] e *Microsoft*[19]. Em todos os casos há uma preocupação com a questão desempenho devido à sobrecarga imposta pelo MMV. A IBM conseguiu resolver este problema no *mainframe* alterando a forma como o *hardware* conversava com o *software*, visto que era a fabricante do *hardware* e *software*. No caso da plataforma *x86*, o *hardware* não estava preparado para isto e os fabricantes do *software* não eram os mesmos do *hardware* (VIRTUOZZO, 2006) (VMWARE, 2005).

Para vencer essa barreira do desempenho surgiu um novo modelo de virtualização: a paravirtualização. A paravirtualização é similar à virtualização completa, exceto que permite a comunicação direta entre o *hardware* e a MV como forma de minimizar o problema de desempenho. Como conseqüência, exige que o SO da MV seja modificado de

[17] *Windows* é o termo genérico utilizado para se referenciar aos sistemas operacionais da família *Windows*, desenvolvida pela Microsoft, para a plataforma *x86*. Exemplos: *Windows 2003, Windows 2008.*
[18] *Parallels* é uma empresa de tecnologia com sede nos Estados Unidos, que oferece soluções de automação e virtualização para estações de trabalho e servidores.
[19] *Microsoft* é uma empresa multinacional de tecnologia com sede nos Estados Unidos, que desenvolve e fabrica uma ampla gama de *softwares* para computadores.

forma a viabilizar esta comunicação. O XEN[20] foi o primeiro produto de mercado que seguiu esse modelo suportando os sistemas operacionais da *Microsoft (Windows)*, e da *Novell*[21] *(linux)* (ROSE, 2004) (BARHAM, 2003).

Recentemente a *Intel* e a *AMD*[22], fabricantes de processadores *x86*, aderiram à nova onda e seus novos processadores já estão saindo com suporte para a virtualização com o intuito de contribuir para a melhoria do desempenho. A *Intel* chamou a sua tecnologia de *VT-x* e a *AMD AMD-v* (OGDEN, 2006).

2.2 - VIRTUALIZAÇÃO EM PLATAFORMA X86

Os desafios em termos de desempenho e suportabilidade da virtualização em plataforma *x86* surgem porque os SO foram projetados para ter total controle sobre o *hardware* e não foram escritos para apoiar a partilha de recursos do *hardware* (HUMPHREYS et al., 2006). Com o advento da virtualização quem passa a monopolizar o *hardware* é a camada de virtualização com o propósito básico de proporcionar a partilha dos dispositivos físicos do *hardware* entre as MV, passando o SO das MV a enxergar somente dispositivos virtuais. Para que o SO continuasse a funcionar perfeitamente foi necessário criar mecanismos de forma ao SO continuar a acreditar que é ele que tem o monopólio do *hardware*. Esses mecanismos interferiram no âmbito da proteção ao SO que conjuga modos de acesso à UCP, anéis de proteção e elementos de interface (CARISSIMI, 2008).

2.2.1 - Componentes de proteção ao SO

Esse capítulo aborda os componentes envolvidos na proteção de um SO, oferece uma visão desses componentes no ambiente tradicional e apresenta a forma como eles foram ajustados para atender as necessidades de uma solução virtualizada. O entendimento dos impactos sofridos por esses ajustes em uma solução virtualizada contribuem para uma melhor visibilidade e entendimento quanto à natural e conseguinte sobrecarga inerente à camada de virtualização.

[20] O *XEN* é um monitor de máquina virtual para plataforma *x86* que utiliza a técnica de paravirtualização.
[21] A *Novell* é uma empresa americana de *software*, especializada em tecnologias de redes, *internet* e sistema operacional *linux.*
[22] A *AMD* é uma multinacional americana de semicondutores, que desenvolve processadores de computadores e tecnologias relacionadas.

2.2.1.1 - Modos de acesso à UCP

Os modos de acesso à UCP têm por objetivo fornecer proteção em caso de violação acidental ou deliberada corrupção do ambiente por algum *software* ou aplicativo. Os modos possíveis são o *kernel* e o usuário.

No modo *kernel* a execução de um código tem acesso completo e irrestrito ao *hardware*, ou seja, para um código executando nesse modo, todos os recursos internos da UCP, tais como registradores e portas de E/S, e áreas de memória podem ser acessados. O modo *kernel* é geralmente reservado para as funções mais confiáveis do SO. Falhas no modo *kernel* são catastróficas e normalmente travam ou interrompem o funcionamento total de um servidor (ATWOOD, 2008).

Os três tipos de eventos que podem fazer um sistema entrar em modo *kernel* são: (i) Interrupções de *hardware*: são modificações no fluxo de controle causadas por uma ação externa, geralmente relacionada a E/S. É um sinal de controle enviado por um dispositivo à UCP, quando um determinado evento é detectado; (ii) Interrupções de *software* ou *traps*: são decorrentes de chamadas de sistemas feitas pelos aplicativos para execução de alguma instrução ou acesso a algum dispositivo que só pode ser feito em modo *kernel*; (iii) Exceções: são operações ilegais causadas por programas.

No modo usuário somente um subconjunto das instruções da UCP, registradores e portas de E/S estão disponíveis. Instruções perigosas como *halt* (parar o processador) e *reset* (reiniciar o processador) são proibidas para todo código executando neste nível. Além disso, o *hardware* restringe o uso da memória, permitindo o acesso somente a áreas previamente definidas. Caso o código em execução tente executar uma instrução proibida ou acessar uma área de memória inacessível, o *hardware* irá gerar uma exceção, desviando a execução para uma rotina de tratamento dentro do *kernel*, que provavelmente irá abortar o programa em execução. Os códigos das aplicações e de algumas funções do SO (que não as mais confiáveis) são executados em modo usuário. Por causa das restrições de acesso à memória e dispositivos de E/S, o modo de usuário quase nada pode fazer sem passar pelo *kernel*. Não pode abrir arquivos, enviar pacotes de rede, imprimir na tela, ou alocar a memória (MAZIERO, 2008).

O modo usuário e *kernel* são providenciais para a estabilidade do sistema computacional, contudo, a mudança do modo usuário para *kernel*, também chamada de

troca de contexto, vem com um alto custo. Esse alto custo ocorre porque, muitas vezes, a quantidade de recursos computacionais manipulados para viabilizar uma troca de contextos é maior que a quantidade de recursos deslocados e alocados para os processos que estão deixando e passando a usar o modo *kernel*, respectivamente. Esse é o motivo pelo qual aplicativos que causam muitas interrupções, por exemplo, normalmente são também muito lentos porque interrupções implicam em transições para o modo *kernel* (ATWOOD, 2008).

2.2.1.2 - Anéis de proteção

Os anéis de proteção, diferentemente dos modos *kernel* e usuário que fazem a proteção do SO por violação de algum *software* ou aplicativo, são usados como mecanismo para proteger os dados e as funcionalidades do sistema computacional contra falhas, violações acidentais e ataques maliciosos. Os anéis são organizados em uma hierarquia da mais privilegiada (geralmente numerada de zero) para a menos privilegiada (geralmente com o maior número de anel) (FAWZI, 2009).

A arquitetura *x86* provê quatro anéis de proteção. O Anel 0 foi projetado para abrigar o *kernel* do SO, o Anel 1 e 2 os *drivers* de dispositivos e o Anel 3 as aplicações de usuários. Os SO que atualmente são executados no *x86* têm utilizado somente o Anel 0 e Anel 3 para conter o *kernel* do SO e as aplicações de usuários, respectivamente (CAMPBELL, et al., 2006). Nesses casos a localização dos *drivers* de dispositivos dependerá muito do SO que pode optar pela execução no modo *kernel* para máxima segurança ou no modo usuário para máxima estabilidade, ou um misto desses dois (ATWOOD, 2008).

O *hardware* restringe severamente as formas em que o controle pode ser passado de um anel para o outro, e também impõe restrições sobre os tipos de acesso à memória que podem ser realizadas através de anéis. Se uma aplicação de usuário tentar executar uma instrução privilegiada ocorrerá uma interrupção que deverá ser tratada adequadamente. Portas especiais entre os anéis são fornecidas para permitir que um anel externo possa acessar os recursos de um anel interno de uma maneira pré-definida, controlada e segura, ao invés de permitir o uso arbitrário (FAWZI, 2009).

2.2.1.3 - Elementos de interface

Em relação aos elementos de interface, um sistema de computação oferece quatro tipos (CARISSIMI, 2008):

- Interface aplicativa de programação: é o meio pelo qual as aplicações acionam uma função de uma biblioteca em nível de usuário para que um serviço de SO possa ser executado;

- Chamadas de sistema: podem ser entendidas como uma porta de entrada para acesso ao *kernel* do SO e aos seus serviços. Sempre que um usuário ou uma aplicação necessita de algum serviço do SO, é realizada uma chamada a uma de suas rotinas. Através dos parâmetros fornecidos na chamada de sistema, a solicitação é processada e uma resposta é enviada à aplicação juntamente com um estado de conclusão indicando o sucesso ou não da operação. Para cada serviço disponível existe uma chamada de sistema associada, e cada sistema operacional possui seu próprio conjunto de chamadas, com nomes, parâmetros e formas de ativação específicos (SILVA, 2010).

- Instruções privilegiadas: são as instruções utilizadas apenas por programas com privilégios especiais para manipular os recursos de *hardware*, sendo o caso do *kernel* dos SO;

- Instruções não privilegiadas: são as instruções utilizadas por qualquer hierarquia acima da UCP, incluindo o SO, componentes não *kernel*. Elas possibilitam que uma aplicação execute um seleto conjunto de instruções, mas não permitem o acesso aos recursos de *hardware* que são, necessariamente, acessados somente através das instruções privilegiadas.

2.2.2 - Arquitetura da Computação Tradicional

No modelo tradicional o SO é o programa mestre com o mais alto nível de privilégio e tem por propósito gerenciar o *hardware* físico e distribuir os dispositivos e componentes físicos aos aplicativos e serviços que deles necessitarem. A Figura 2.1 mostra a relação entre as camadas de um computador tradicional onde um único SO monopoliza todo o *hardware* (DESAI, 2007).

Nesse modelo os modos de acesso à UCP (*kernel* e usuário), os anéis de proteção (0 e 3) e os elementos de interfaces, operam conforme exposto. Ou seja, e conforme demonstrado na Figura 2.2, a aplicação reside no Anel 3, o SO no Anel 0, as instruções não privilegiadas da aplicação são diretamente tratadas entre a aplicação e a UCP, e a execução

das instruções privilegiadas exigem as chamadas de sistemas para que o *kernel* do SO possa executá-las.

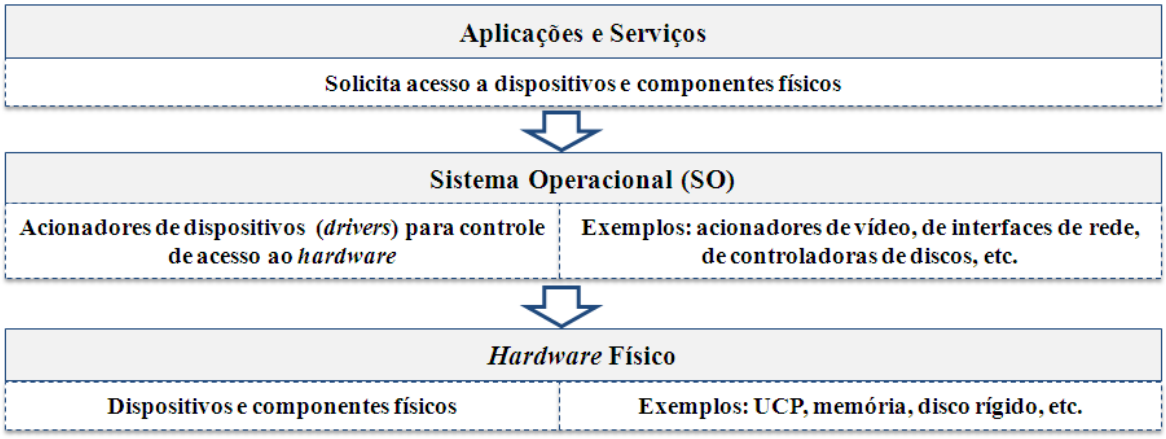


Figura 2.1 – Visão geral da relação direta entre sistema operacional e *hardware* (DESAI, 2007)

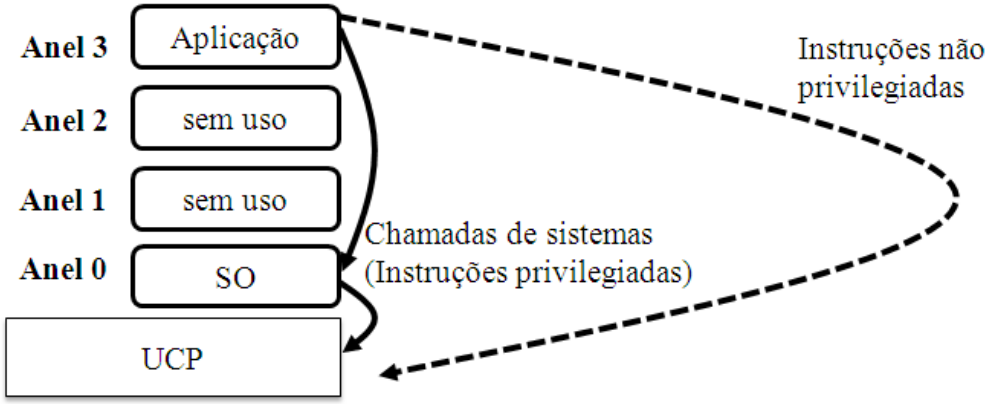


Figura 2.2 – Visão geral da computação tradicional (modificada - VMWARE, 2007)

2.2.3 - Arquitetura da Solução Virtualizada

Em uma solução de virtualização quem passa a ser o programa mestre com o mais alto nível de privilégio é a camada de virtualização, também chamada de MMV. Como consequência, a camada de virtualização passa a monopolizar todo o *hardware*, e recursos importantes como UCP, memória e disco rígido poderão então servir a muitas MV, tendo cada MV seu próprio SO, aplicações e serviços conforme demonstrado na Figura 2.3.

Nessa arquitetura, os recursos físicos do servidor são entregues para as MV como dispositivos virtuais, tais como UCP virtual, memória virtual e disco virtual. O SO instalado em uma MV passa a ser chamado de SO hóspede. As versões específicas de SO a serem instaladas em uma MV irão variar em função da solução de virtualização, mas quase sempre a meta será de dar suporte ao maior número possível de tipos de SO (DESAI, 2007).

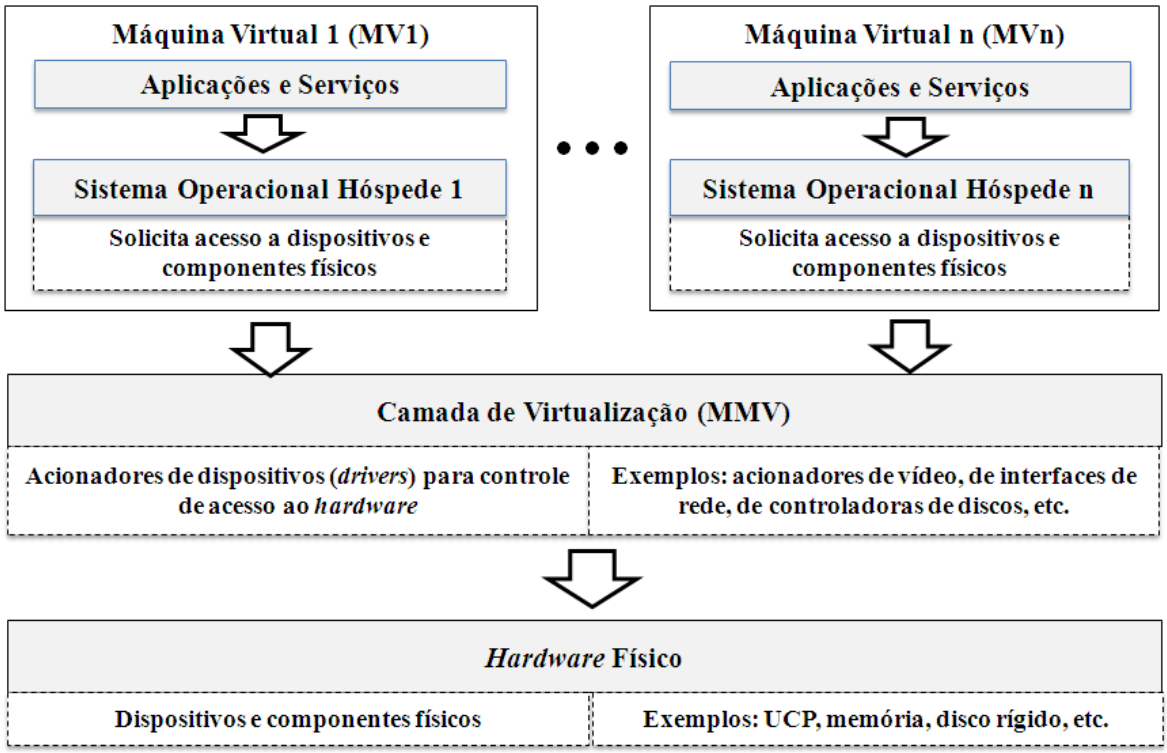


Figura 2.3 – Visão geral da virtualização aplicada a servidores (modificado - MENASCÉ, 2005)

O SO hóspede e os aplicativos instalados em uma MV normalmente não sabem que estão executando em um ambiente virtual, embora eles usem os recursos do *hardware* físico como se estivessem em um computador físico isolado. A administração dessa forma isolada de apresentar recursos para as MV provocou o surgimento de uma nova classe de instruções conforme a sua sensibilidade em afetar ou não outra MV, as chamadas instruções sensíveis e não sensíveis, respectivamente.

As instruções sensíveis são instruções que em um contexto de virtualização podem interferir na execução de outros SO que compartilham os recursos de *hardware*, comprometendo o isolamento entre os SO hóspedes. Por exemplo: instruções de acesso ao

registrador da tabela de páginas de memória. Essas instruções devem ser detectadas pela solução de virtualização que deve tratá-las de maneira a não comprometer o isolamento.

As instruções não sensíveis não comprometem o isolamento entre os SO hóspedes, podendo ser executadas diretamente no *hardware*, sem a interferência da solução de virtualização.

A grande maioria das instruções sensíveis são também instruções privilegiadas, isso facilita o monitoramento feito pelo MMV, pois ele só precisa conferir as instruções que geraram uma interrupção no SO hóspede para identificar se elas são sensíveis ou não.

Para normatizar as características de um MMV, em 1974, Popek e Goldberg (POPEK,, et al., 1974) introduziram três propriedades necessárias para que um sistema computacional de virtualização possa ser considerado um MMV:

- Controle de recursos (segurança): um MMV deve ter controle completo sobre os recursos virtualizados sendo estritamente proibido que um programa executando sobre a MV os acesse diretamente.

- Equivalência de execução (fidelidade): um programa executando sobre uma MV deve exibir um comportamento idêntico àquele apresentado caso a MV não existisse e o programa acessasse diretamente uma máquina física equivalente. Duas exceções são consideradas. Primeira, eventualmente, algumas instruções podem ter seu tempo de execução aumentado. Segunda, pode haver problemas de conflito de acesso a recursos, os quais devem ser resolvidos de forma apropriada pelo MMV.

- Eficiência (desempenho): todas as instruções de máquina que não comprometem o funcionamento do sistema, as instruções não sensíveis, devem ser executadas diretamente no *hardware* sem intervenção do MMV. Para atender esse quesito, é permitido ao MMV utilizar uma técnica chamada de execução direta que simplifica a implementação do MMV e aumenta o desempenho. Com a execução direta, o MMV configura a UCP em um modo com privilégios reduzidos de maneira tal que o SO hóspede não pode executar diretamente suas instruções sensíveis (privilegiadas). A execução com privilégios reduzidos gera interrupções, por exemplo, quando o SO hóspede tenta chamar uma instrução sensível. O MMV precisa, então, de corretamente tratar as interrupções para permitir a correta

execução do SO hóspede na MV. Especificamente, ele executa o SO hóspede na MV com privilégios reduzidos de maneira tal que o efeito de qualquer sequência de instrução é garantido ser contido somente na MV. Em função disso, o MMV deve manusear somente as interrupções que resultam das tentativas da MV executar instruções privilegiadas (LIM, 2004). A Figura 2.4 retrata o fluxo de instruções com execução direta entre o SO e a UCP em um sistema tradicional e um com virtualização

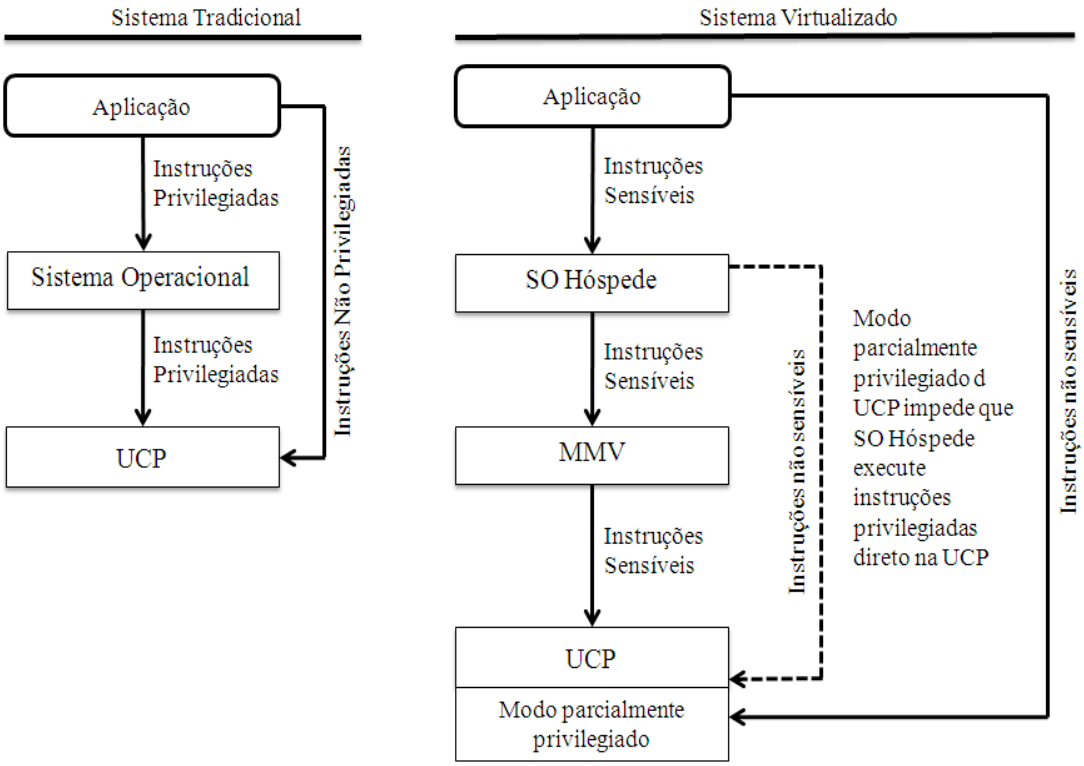


Figura 2.4 – Interação SO x UCP no sistema tradicional e virtualizado

Um MMV pode ser implementado por *software*, assistido por *hardware*, ou um misto desses dois. Os MMV via *software* antecederam aos MMV assistidos por *hardware* e baseiam-se na redução de privilégios do SO hóspede, e o MMV assistido por *hardware* pretende resgatar os privilégios do SO hóspede.

2.2.3.1 - MMV com implementação via *software*

Os MMV via *software* baseiam-se na redução de privilégios do SO hóspede, passando o MMV a executar no Anel 0, o SO hóspede no Anel 1 ou 2 (no modelo tradicional era no Anel 0) e as aplicações no Anel 3 (não houve alterações). Como

consequência desse reposicionamento dos anéis, a camada de virtualização passa a ser, essencialmente, o manipulador de falhas, pois as operações sensíveis executadas pelo SO hóspede geram interrupções, que por sua vez são emuladas ou executadas conforme a solução de virtualização. Essa manipulação implica em: (i) interpretação e tradução de instruções; (ii) interpretação das instruções de UCP para o caso em que a MV não tem a mesma arquitetura do servidor; (iii) tradução das instruções para seu formato nativo e execução (ZOVI, 2006). Esta abordagem funciona bem, entretanto gera vários desafios relativos a desempenho. Excesso de interrupções, imitação do Anel 1 para operar como Anel 0 e excesso de troca de contextos de UCP são alguns deles (HUMPHREYS, et al., 2006).

Quando uma solução de virtualização se utiliza da emulação para executar uma interrupção gerada por uma instrução sensível diz-se que ela está utilizando a técnica de virtualização completa. Nessa técnica a reescrita do código da instrução privilegiada ocorre de forma dinâmica, em tempo de execução, pois as interfaces apresentadas pelo MMV para as MV são as mesmas do servidor físico, havendo uma replicação virtual de toda a arquitetura do *hardware*. A Figura 2.5 ilustra a topologia da virtualização completa.

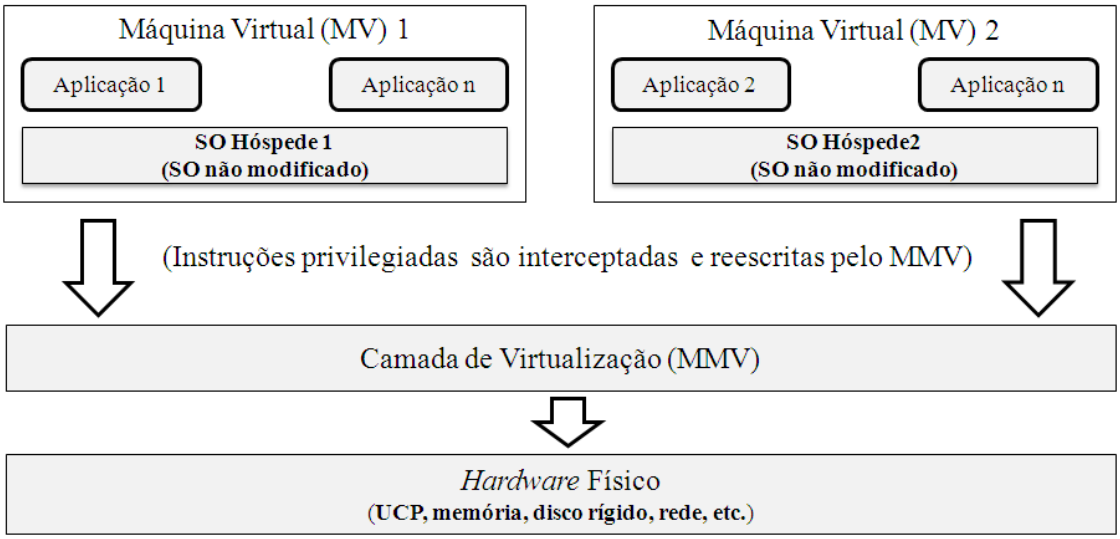


Figura 2.5 – Topologia da virtualização completa (modificada - MENASCÉ, 2005)

Nessa topologia (virtualização completa), para toda instrução do SO hóspede encaminhada à UCP há a interceptação do MMV para apurar se ela é ou não uma instrução sensível que deve ser executada somente pela camada de virtualização. A tabela Tabela 2.1 resume as principais vantagens e desvantagens inerentes à virtualização completa.

Tabela 2.1 – Vantagens e desvantagens da virtualização completa

Vantagens	**Desvantagens**
Executa SO hóspede não modificado	Desempenho (necessidade de tratamento de todas as operações privilegiadas)
Oferece uma réplica do *hardware*	
Provê maior compatibilidade arquitetural entre diferentes arquiteturas de UCP	
SO hóspede não sabe da existência da virtualização	

Quando uma solução de virtualização se utiliza da execução direta a técnica utilizada é a paravirtualização e a reescrita do código da instrução privilegiada ocorre de forma estática e para isso, o SO hóspede precisa ser modificado (CARISSIMI, 2008). A paravirtualização difere da virtualização completa porque ela não emula recursos de *hardware* para execução das tarefas privilegiadas, mas oferece uma interface onde os SO hóspedes solicitam diretamente ao MMV a execução das instruções privilegiadas, evitando, assim, a necessidade da reescrita do código em tempo de execução (KOSLOVSKI, et al., 2006). A Figura 2.6 exibe a topologia da virtualização paravirtualizada.

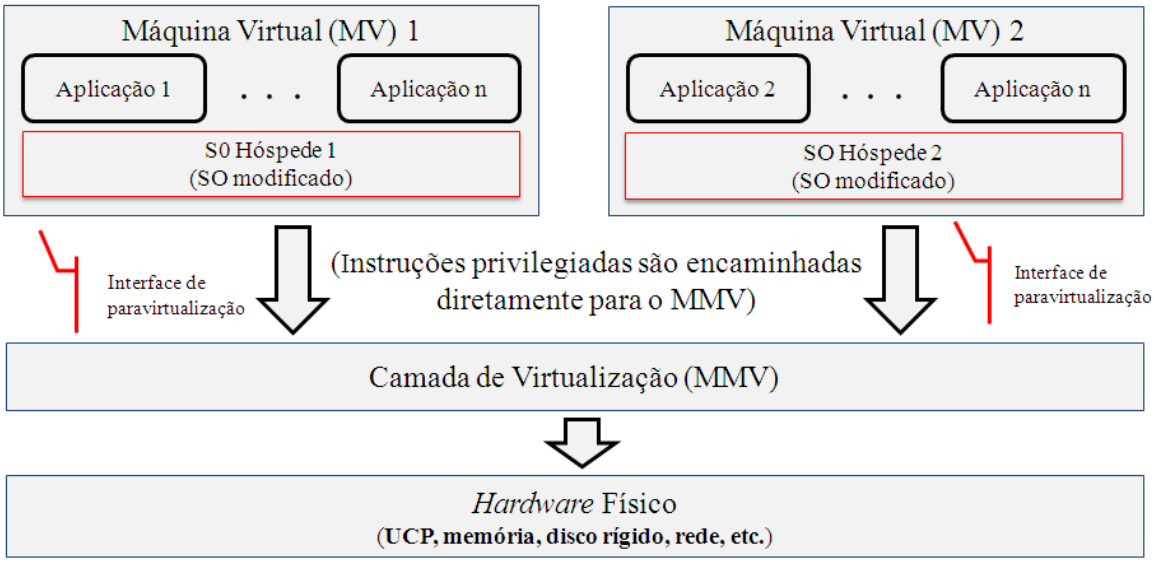


Figura 2.6 – Topologia da paravirtualização (modificada - VMWARE, 2007)

A paravirtualização, quando aplicada em arquiteturas que não oferecem assistência à virtualização via *hardware* (a seguir descrito), dependendo do perfil da aplicação pode resultar em melhor desempenho se comparada à virtualização completa onde a camada de

virtualização precisa interferir constantemente quando são executadas instruções sensíveis (FERREIRA, 2008). A Tabela 2.2 a seguir resume as principais vantagens e desvantagens relativas à paravirtualização.

Tabela 2.2 – Vantagens e desvantagens da paravirtualização

Vantagens	**Desvantagens**
Elimina ou reduz a necessidade da reescrita do código em tempo de execução	SO hóspede tem consciência da existência da virtualização
	Requer modificação nos SO hóspedes

2.2.3.2 - MMV com implementação via *hardware*

Por volta de 2005, os principais fabricantes de processadores *x86*, *Intel* e *AMD*, incorporaram um suporte básico à virtualização em suas UCP. As soluções da *Intel* e da *AMD* foram desenvolvidas independentemente uma da outra e são incompatíveis, embora sirvam para o mesmo propósito e conceitualmente sejam equivalentes (CARISSIMI, 2008).

A virtualização via *hardware* exige compatibilidade de *hardware*, processador de 64 *bits* para solução da AMD e 32 e 64 *bits* para solução da *Intel*. Basicamente a virtualização assistida por *hardware* pretende atuar sobre os seguintes aspectos (HUMPHREYS, et al., 2006):

- Menor complexidade. As soluções de virtualização via *software* exigem o uso do MMV para que interrupções privilegiadas ou sensíveis geradas pelo SO hóspede sejam executadas, o que exige o uso da reescrita dinâmica ou estática no SO hóspede para que o SO hóspede acredite que é ele que está executando a instrução. A virtualização assistida por *hardware* pretende eliminar a necessidade da reescrita dinâmica ou estática.

- Aumento do desempenho da solução de virtualização reduzindo o processamento do MMV onde não será mais necessário fazer a tradução das operações de E/S, de gerenciamento de memória e de simulação das instruções privilegiadas.

- Maior confiabilidade. Há uma preocupação da reescrita dinâmica ou a modificação do SO hóspede reduzir a confiabilidade da solução de virtualização. A virtualização

assistida por *hardware* aborda esta questão através do suporte de *hardware* para expansão do privilégio de anel, e por conseqüência simplificação do MMV. Privilégio de expansão do anel significa que o MMV é executado em um novo anel de privilégio, permitindo assim que o SO hóspede possa ser executado em seu anel de privilégio nativo (Anel 0).

A idéia central das tecnologias da *Intel* e *AMD* para a plataforma *x86* consiste em definir dois modos possíveis de operação do processador (UCP): os modos *root* e *non-root.* O modo *root* se destina à execução de um MMV. O modo *non-root* à execução de MV. Ambos os modos suportam os quatro níveis de privilégio (anéis 0, 1, 2, 3). O MMV é executado no Anel 0 no modo *root* e o SO hóspede no Anel 0 do modo *non-root.* Neste contexto, o SO hóspede passa a não precisar da reescrita dinâmica de seu código (MAZIERO, 2008).

São também definidos dois procedimentos de transição entre modos: *VM entry* (transição *root*→ *non-root*) e *VM exit* (transição *non-root*→ *root).* Quando operando dentro de uma MV (ou seja, em modo *non-root*), as instruções sensíveis e as interrupções podem provocar a transição *VM exit*, devolvendo o processador ao MMV em modo *root.* As instruções e interrupções que provocam a transição *VM exit* são configuráveis pelo próprio MMV. Para gerenciar o estado do processador (conteúdo dos registradores), é definida uma Estrutura de Controle de Máquina Virtual (em inglês *Virtual-Machine Control Structure* - *VMCS*). Essa estrutura de dados contém duas áreas: uma para os SO hóspedes e outra para o MMV. Na transição *VM entry*, o estado do processador é lido a partir da área de SO hóspedes da *VMCS*. Já uma transição *VM exit* faz com que o estado do processador seja salvo na área de SO hóspede e o estado anterior do MMV, previamente salvo na *VMCS*, seja restaurado. A Figura 2.7 traz uma visão geral da arquitetura implementada pela *Intel* para o *VMCS* (MAZIERO, 2008).

O suporte de *hardware* necessário para a execução de um MMV eficiente encontra-se implementado de forma parcial nos processadores *x86.* Por exemplo, a família de processadores *Intel Pentium IV*[23] (e anteriores) possui 17 instruções sensíveis que podem ser executadas em modo usuário sem gerar interrupções, o que viola as propriedades de Popek e Goldberg e dificulta a criação de MV em sistemas que usam esses processadores (MAZIERO, 2008).

[23] O *Intel Pentium IV* é a quinta geração de microprocessadores com arquitetura *x86* fabricados pela *Intel.*

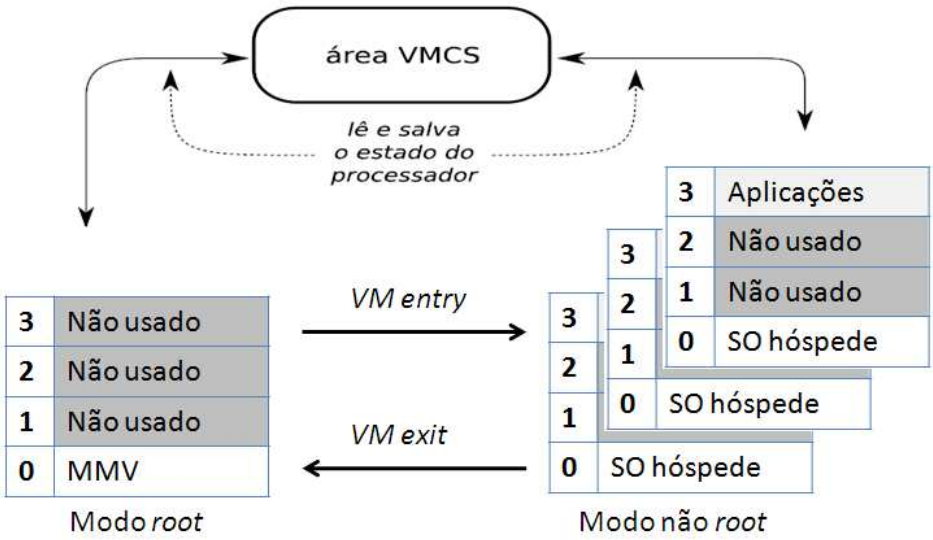


Figura 2.7 – Visão geral da arquitetura Intel para o *VMCS* (MAZIERO, 2008)

Adicionalmente, os sistemas virtualizados utilizando somente técnicas de *hardware* vêem apresentando desempenho inferior aos sistemas virtualizados que utilizam apenas técnicas de *software*. Uma possível causa, segundo Ferreira (FERREIRA, 2008), pode ser a utilização incorreta do suporte de *hardware* e outra o suporte de *hardware* oferecido pelos processadores para a virtualização que não é utilizado em conjunto com as já bem estabelecidas técnicas de *software*.

A virtualização assistida por *hardware* não sobrepõe a via *software* e MMV mais recentes vêm ofertando flexibilidade para manualmente se optar pelo tipo de virtualização que deverá ser atribuído à uma MV ou deixar que o próprio MMV decida pelo melhor modo. Esta facilidade se deve ao fato de que nem todos os modos provêem desempenho similar e o desempenho a ser atingido por uma aplicação dependerá muito do seu perfil (VMWARE, 2008).

3 - ESTUDO COMPARATIVO

Este capítulo descreve as características da carga de trabalho, dos *hardwares*, dos *softwares*, e da metodologia aplicada aos testes, cujo objetivo visa identificar a sobrecarga de processamento na camada de banco de dados decorrente da virtualização, para os quesitos tempo de resposta e consumo de recursos computacionais. As técnicas de virtualização cobertas pelos testes foram a virtualização completa sem assistência por *hardware*, a paravirtualização e a virtualização completa assistida por *hardware*.

3.1 - CARACTERÍSTICAS DA CARGA DE TRABALHO

A carga de trabalho usada nos testes foi gerada através da ferramenta *Swingbench*, versão 2.3, desenvolvida por *Dominic Giles* do Grupo de Soluções de Banco de Dados da *Oracle*, para fins de demonstração, testes e *benchmarks*. Tem por propósito provocar cargas em bancos de dados *Oracle* e exercitar o consumo de recursos do servidor. Sua arquitetura é baseada no modelo cliente/servidor, conforme Figura 3.8 (PORTAL DOMINIC GILES, 2010).

As transações comandadas pelo Cliente *Swingbench* para o Banco de Dados *Swingbench* utilizam o *driver JDBC*. Essas transações são geradas através de aberturas de sessões no banco de dados conforme a quantidade de usuários simultâneos configurados no Cliente *Swingbench*. Após a execução de um teste é emitido um relatório com informações resultantes do teste, tais como total de transações processadas, tempo de resposta obtido, consumo de UCP para os modos usuários e *kernel*, dentre outros (PORTAL DOMINIC GILES, 2010).

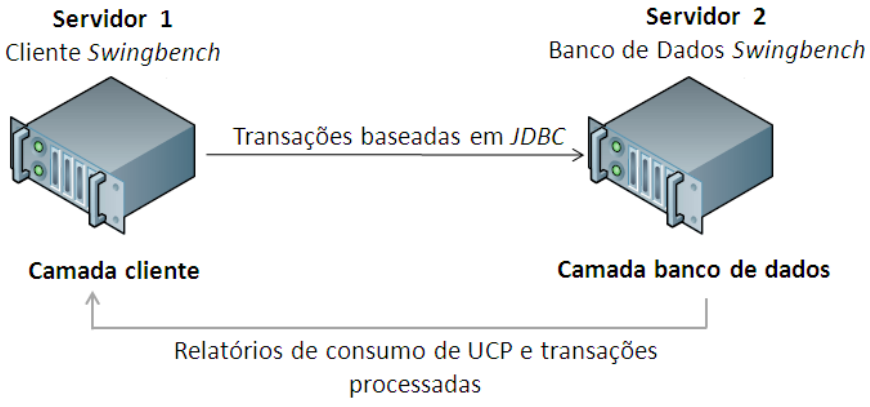


Figura 3.8 – Arquitetura do *Swingbench* com *benchmark Order Entry*

O *Swingbench* versão 2.3 suporta três diferentes tipos de *benchmarks*: *Calling Circle, Sales History e Order Entry.* O *Calling Circle* foi projetado para testes simples de transações *OLTP*, o *Sales History* para testes de *DSS*[24] e o *Order Entry* para emulação de transações clássicas *OLTP* com taxas de E/S mais intensas. O *Order Entry* foi o *benchmark* utilizado em função de sua característica de E/S intensos, inerentes aos bancos de dados das grandes empresas de telecomunicações. Ele modela a clássica transação de tratamento de pedidos, com muitas e pequenas transações, e com leituras e escritas randômicas. É constituído pelas cinco transações típicas de uma aplicação que trata entradas de pedidos:

- Criação de um cliente, chamada *Customer Registration;*
- Visualização de produtos, chamada *Browse Products*;
- Cadastramento de pedidos de produtos, chamada *Orders Products;*
- Processamento de pedidos, chamada de *Process Orders*;
- Visualização de pedidos, chamada *Browse Orders.*

O banco de dados do *Order Entry,* instância *SOE,* foi configurado para tratar um milhão de clientes e um milhão de pedidos. Nesta configuração o *Swingbench* cria, automaticamente, um banco de dados com consumo de área em disco de 3,7GB, e constituído pelas seguintes tabelas:

Tabela 3.3 – Detalhes das tabelas do banco de dados

Nome da Tabela	**Quantidade de Linhas**	**Quantidade de blocos**	**Tamanho em bytes**
WAREHOUSES	20	10	106.496
PRODUCT_INFORMATION	288	10	106.496
PRODUCT_DESCRIPTIONS	288	22	212.992
CUSTOMERS	1.000.000	9.440	80.084.992
ORDER_ITEMS	3.501.144	11.059	93.609.984
ORDERS	1.000.000	4.544	39.190.528
INVENTORIES	5.760	828	7.561.216
LOGON	0	0	1.703.936

[24] *DSS, Decision Support System*, é um sistema de suporte aos negócios e às decisões de uma organização. Um sistema *DSS* tipicamente é um sistema interativo, que ajuda na compilação de dados, documentos, conhecimento e modelos de negócios de forma a se identificar e solucionar problemas, e tomar decisões.

A opção pelo tamanho do banco de dados em 3,7GB foi decorrente da premissa de não utilizar área maior que 40% da capacidade do disco de banco de dados, que possuía 9,7GB, com o objetivo de evitar contenção de E/S. Um número maior de clientes e pedidos implicaria em extrapolar essa condição. Além disso, a volumetria de um milhão de clientes e um milhão de pedidos é encontrada no mundo real das aplicações implementadas em grandes empresas de telecomunicações, incluindo a empresa onde esses testes foram realizados.

A taxa de E/S configurada foi de 40% para leitura e 60% para escrita de forma a refletir um comportamento tipicamente *OLTP*[25]. Um comportamento com taxa de leitura superior à de escrita pende para um perfil *OLAP*[26], cujas soluções de *hardware* e *software* comumente são proprietárias (FELBER, 2005), não se aplicando, desta forma, ao ambiente virtualizado.

3.2 - CONFIGURAÇÃO DO *HARDWARE*

O *hardware* utilizado para os testes constituiu-se de dois servidores físicos, com configurações idênticas: servidor *x86*, modelo *Dell Power Edge* 1950, um processador *Intel Xeon* E5410 @ 2.33 *Ghz* com quatro *cores*, trinta e dois *GB* de memória *RAM*, uma controladora *PERC* 6i com dois discos internos *SAS* de *146 GB* 15.000 *RPM*, e tecnologia de virtualização via *hardware* para UCP (*Intel VT-x*) e memória (*EPT*).

Devido à característica cliente/servidor do *Swingbench*, um dos servidores foi utilizado como servidor cliente e o outro como servidor de banco de dados obedecendo a topologia detalhada na Figura 3.9.

[25] *OLTP* é um acrônimo de *On-Line Transaction Processing*, ou Processamento de Transações em Tempo Real. Aplica-se aos sistemas categorizados como sistemas de informações tradicionais, que se encarregam de registrar as transações contidas em uma determinada operação organizacional.
[26] *OLAP* é um acrônimo de *On-Line Analytical Processing*, ou Processamento Analítico em Tempo Real. Aplica-se aos sistemas com capacidade para manipular e analisar um grande volume de dados sob múltiplas perspectivas. Essas aplicações são usadas pelos gestores em qualquer nível da organização para análises comparativas que facilitem a tomada de decisões diárias.

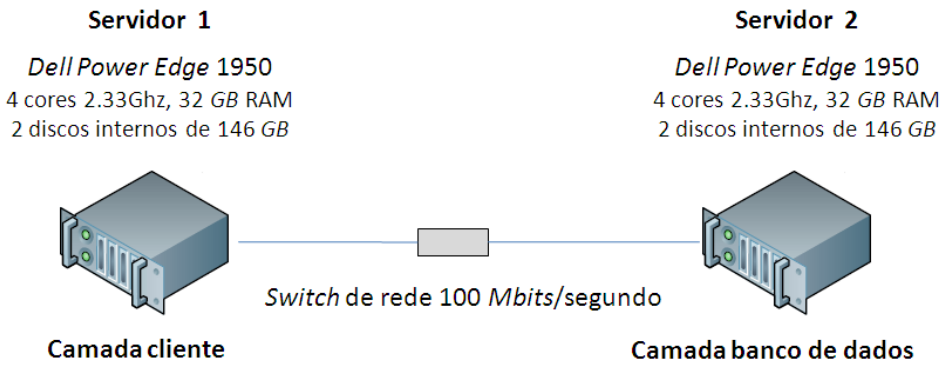


Figura 3.9 – Topologia da infraestrutura de testes

3.3 - SOFTWARES

Os componentes de *softwares* utilizados nos testes foram:

- MMV *VMware ESXi 4.0.0 build-2081167 com* acesso remoto via *vSphere Client*, do fornecedor *VMware*. O binário do *VMware ESXi* pode ser obtido na internet através do sítio (PORTAL VMWARE, 2010);

- Sistema Operacional *Red Hat Enterprise Linux Server* (LRH) na versão 5, 32 *bits*, fornecedor *Red Hat*. A opção pelo LRH 32 *bits* foi necessária porque na tabela de compatibilidade do *VMware ESXi* 4.0 com processador *Intel Xeon* E5410 há o suporte do SO LRH somente para 32 *bits* (VMWARE, 2008). O binário do LRH utilizado pode ser obtido através do sítio (PORTAL RED HAT, 2010);

- Sistema de Gestão de Base de Dados (SGBD) *Oracle 11G R* 2, da Oracle, com binário de instalação disponível no sítio (PORTAL ORACLE, 2010);

- *Swingbench* versão 2.3, com binário de instalação disponível no sítio (PORTAL DOMINIC GILES, 2010).

3.4 - METODOLOGIA

O estudo comparativo foi conduzido em uma abordagem que compreendeu realizar testes em idênticas configurações de *hardware* no ambiente tradicional e virtualizado, de forma com que os resultados obtidos no servidor tradicional fossem comparados com o

ambiente virtualizado e, por conseguinte, fosse possível identificar e calcular a sobrecarga decorrente da virtualização.

Os testes tiveram como escopo injetar a mesma carga de trabalho, em mesmo período de tempo, em um servidor tradicional e em ambiente virtualizado, ambos atendendo a camada de banco de dados. O servidor tradicional representou o ponto ideal (ótimo) em relação aos resultados obtidos para os quesitos tempo de resposta e consumo de recursos do servidor visto que não foi impactado por uma camada de virtualização. Desta forma, os resultados obtidos pelo servidor tradicional serviram de referência e parâmetro para calcular a sobrecarga decorrente da virtualização.

Os cenários montados para aferir a sobrecarga de processamento na camada de banco de dados cobriram as técnicas de virtualização completa sem assistência por *hardware*, paravirtualização e virtualização completa assistida por *hardware*. Para tanto, foram criadas quatro infraestruturas específicas para suportar os testes e coletas de dados:

- Infra1: ambiente tradicional, sem uso da virtualização.
- Infra2: ambiente virtualizado, técnica virtualização completa, sem assistência por *hardware*.
- Infra3: ambiente virtualizado, técnica paravirtualização, sem assistência por *hardware*.
- Infra4: ambiente virtualizado, técnica virtualização completa, com assistência de virtualização por *hardware.*

Devido à característica cliente/servidor do *Swingbench,* os dois servidores físicos atenderam a essas quatro infraestruturas de forma serializada, obedecendo a seguinte sequência: Infra1, Infra2, Infra3, Infra4. A Tabela 3.4 relaciona as funções e *softwares* usados nesses dois servidores para cada infraestrutura coberta, e atribui um codinome para os servidores de banco de dados para facilitar a sua identificação e referência a partir deste ponto. Para a Infra1 o codinome atribuído foi SRV1, para a Infra2 MV1, para a Infra3 MV2 e para a Infra4 MV3.

Tabela 3.4 – Funções e *softwares* dos servidores de testes

Infra	Servidor	Funções e *softwares* instalados no servidor	Classificação do componente	Codinome
Infra1	1	Função: camada cliente *Softwares*: LRH, *Swingbench* e *Oracle*, instalados no primeiro disco interno	Servidor físico	Não se aplica
	2	Função: camada banco de dados *Softwares*: LRH, *Swingbench* e *Oracle*, instalados no primeiro disco interno	Servidor físico	SRV1
Infra2	1	Função: camada cliente *Softwares*: LRH, *Swingbench* e *Oracle*, instalados no primeiro disco interno	Servidor físico	Não se aplica
	2	Função: MMV *Softwares*: *Vmware ESXi* instalado no primeiro disco interno	Solução de virtualização	Não se aplica
		Função: camada banco de dados Softwares: LRH, *Swingbench* e *Oracle*, instalados no segundo disco interno	Máquina virtual	MV1
Infra3	1	Função: camada cliente *Softwares*: LRH, *Swingbench* e *Oracle*, instalados no primeiro disco interno	Servidor físico	Não se aplica
	2	Função: MMV *Softwares*: *Vmware ESXi* instalado no primeiro disco interno	Solução de virtualização	Não se aplica
		Função: camada banco de dados *Softwares*: LRH, *Swingbench* e *Oracle*, instalados no segundo disco interno	Máquina virtual	MV2
Infra4	1	Função: camada cliente *Softwares*: LRH, *Swingbench* e *Oracle*, instalados no primeiro disco interno	Servidor físico	Não se aplica
	2	Função: MMV *Softwares: Vmware ESXi* instalado no primeiro disco interno	Solução de virtualização	Não se aplica
		Função: camada banco de dados *Softwares*: LRH, *Swingbench* e *Oracle*, instalados no segundo disco interno	Máquina virtual	MV3

O servidor *Dell Power Edge* 1950 possui nativamente os recursos de virtualização assistida por *hardware*. Por padrão, esses recursos vêm desabilitados. Para os servidores SRV1, MV1 e MV2 esse recurso foi mantido desabilitado. Para a MV3, esses recursos foram habilitados utilizando o seguinte procedimento: o servidor foi fisicamente ligado, ao aparecer a mensagem "<F2> = *System Setup*", a tecla F2 foi pressionada e o programa de configuração do servidor foi exibido. Foi escolhida a opção "*CPU Information*" e na tela de informações da UCP foi modificado o campo *Virtualization Technology* de *disable* para *enabled*.

Durante a criação da MV1, MV2 e MV3, suas propriedades foram configuradas de forma a atender a técnica de virtualização do teste específico da infraestrutura associada. Maiores detalhes podem ser vistos no Anexo A.

O SRV1 foi projetado para suportar os testes em ambiente tradicional, portanto constituído de uma relação direta entre SO LRH e *hardware* para tratamento das instruções privilegiadas. O SO LRH ficou hospedado no Anel 0 e os comandos e solicitações de acesso ao *hardware* foram executados partindo-se do modo usuário, logo Anel 3. Neste cenário, as instruções não privilegiadas são diretamente tratadas entre aplicação e *hardware* e as instruções privilegiadas requerem a intervenção do *kernel* do SO LRH para serem executadas através das chamadas de sistema ao SO. Os servidores MV1, MV2 e MV3 foram projetados para testes em ambiente virtualizado e por isso acrescentam uma camada de virtualização, o *VMware ESXi* (VMware 2010) (VMWARE, 2008).

A MV1, por utilizar somente a virtualização via *software*, implica no reposicionamento do LRH no Anel 1 para que o *VMware ESXi* ocupe o Anel 0. Neste cenário, toda instrução privilegiada ou sensível é interceptada e executada pelo *VMware ESXi*, pois somente ele tem acesso irrestrito os recursos de *hardware.*

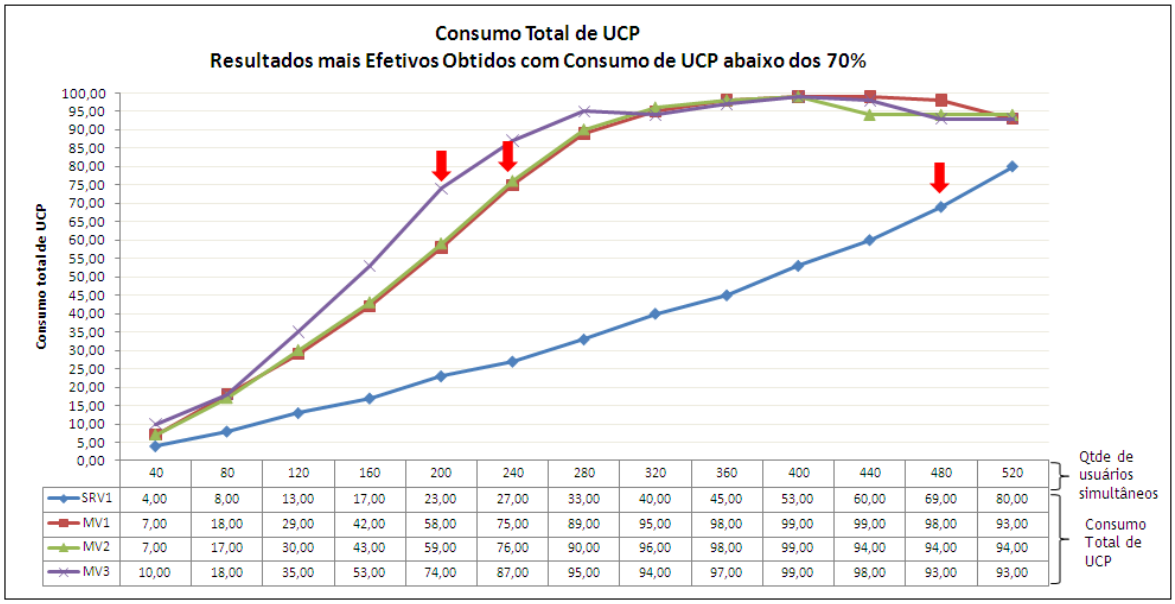
A MV2 utiliza a mesma topologia da MV1, contudo usa também a paravirtualização no qual o SO LRH é modificado para permitir que um conjunto de instruções privilegiadas possa ser executado diretamente entre o SO hóspede e o *hardware*.

Já a MV3, por usar a assistência da virtualização por *hardware,* faz uso dos modos *root* e *non-root,* posicionando o *VMware ESXi* no Anel 0 do *root* e LRH no Anel 0 do *non-root.*

Para a obtenção de resultados mais precisos, as instalações, configurações e parametrizações dos *softwares* envolvidos seguiram as melhores recomendações utilizadas pela empresa de telecomunicações onde esses testes foram realizados e pelos fornecedores dos *softwares* envolvidos. Os Anexos B, C e D relacionam os parâmetros e configurações utilizados para o *VMware ESXi,* LRH *e Oracle,* respectivamente. A criação e configuração da base de dados *Order Entry* ocorreu de forma idêntica em todas as infraestruturas.

Para assegurar o rigor das configurações dos ambientes utilizados nos cenários comparativos, todos os *softwares* foram instalados em disco interno com sistemas de arquivos iguais e de mesmo tamanho em todas as infraestruturas (detalhes no Anexo E). Além disso, todos os testes realizados no SRV1 foram identicamente executados na MV1, MV2 e MV3.

Os testes de carga foram realizados por meio da criação de sessões simultâneas no banco de dados utilizando-se o *Swingbench*. A quantidade de sessões criada no banco de dados depende da quantidade de usuários simultâneos configurados para cada teste. Por recomendação do criador da ferramenta do *Swingbench* (PORTAL DOMINIC GILES, 2010), para resultados mais confiáveis, mais seguros, os testes de carga devem considerar um consumo de UCP no servidor de banco de dados abaixo dos 70%. Testes preliminares executados no SRV1, MV1, MV2 e MV3 permitiram apurar que com 200 usuários simultâneos a MV3 extrapolou os 70%, com 240 extrapolaram a MV1 e MV3 e o SRV1 extrapolou com 480 (Figura 3.10). Com esse resultado, o limite de usuários concorrentes para os testes foi fixado em 180 porque até essa quantidade nenhum dos servidores atingem os 70% de UCP.



	40	80	120	160	200	240	280	320	360	400	440	480	520
SRV1	4,00	8,00	13,00	17,00	23,00	27,00	33,00	40,00	45,00	53,00	60,00	69,00	80,00
MV1	7,00	18,00	29,00	42,00	58,00	75,00	89,00	95,00	98,00	99,00	99,00	98,00	93,00
MV2	7,00	17,00	30,00	43,00	59,00	76,00	90,00	96,00	98,00	99,00	94,00	94,00	94,00
MV3	10,00	18,00	35,00	53,00	74,00	87,00	95,00	94,00	97,00	99,00	98,00	93,00	93,00

Figura 3.10 – Teste para identificação do número máximo de usuários simultâneos

Para uma melhor percepção quanto à sobrecarga imposta pela camada de virtualização, os testes tiveram duração de 10 minutos cada e consideraram o aumento progressivo da quantidade de usuários simultâneos abrindo sessões no banco de dados. As quantidades de usuários simultâneos utilizados nos testes foram: 20, 40, 60, 80, 100, 120,

140, 160 e 180. Para corroborar com o rigor e precisão dos testes, no início de cada teste era recriada a instância SOE e reinicializado o SGBD *Oracle*.

As ferramentas utilizadas para coletar dados do consumo de recursos do servidor de banco de dados durante a realização dos testes foram os nativos do *Swingbench* e do LRH, a saber:

- Relatório gerado pelo *Swingbench;*
- Estatísticas geradas pelo utilitário *iostat* do LRH sobre utilização de processadores e operações de E/S para dispositivos e sistemas de arquivos;
- Estatísticas geradas pelo utilitário *vmstat* do LRH sobre processos, processadores, E/S, paginação e memória;
- Estatísticas geradas pelo utilitário *sar* do LRH sobre as atividades do sistema, incluindo E/S, paginação, processos, interrupções, rede e processadores.

4 - RESULTADOS E DISCUSSÃO

Neste capítulo são analisados os resultados obtidos nos testes realizados no ambiente tradicional (SRV1) e nos ambientes virtualizados (MV1, MV2 e MV3), numa abordagem que compreende comparar os resultados apurados. Os objetos de análise e comparação são o total de transações processados, os tempos de respostas obtidos nas cinco transações do *benchmark Order Entry (Customer Registration, Browse Product, Order Product, Process Order e Browse Order)* e os consumos de recursos computacionais. Os consumos de recursos computacionais examinados neste estudo foram UCP, memória, acesso a disco (leitura e escrita), processos e segmentos, interrupções e trocas de contextos.

4.1 - COMPARATIVOS DE TRANSAÇÕES EM BANCO DE DADOS

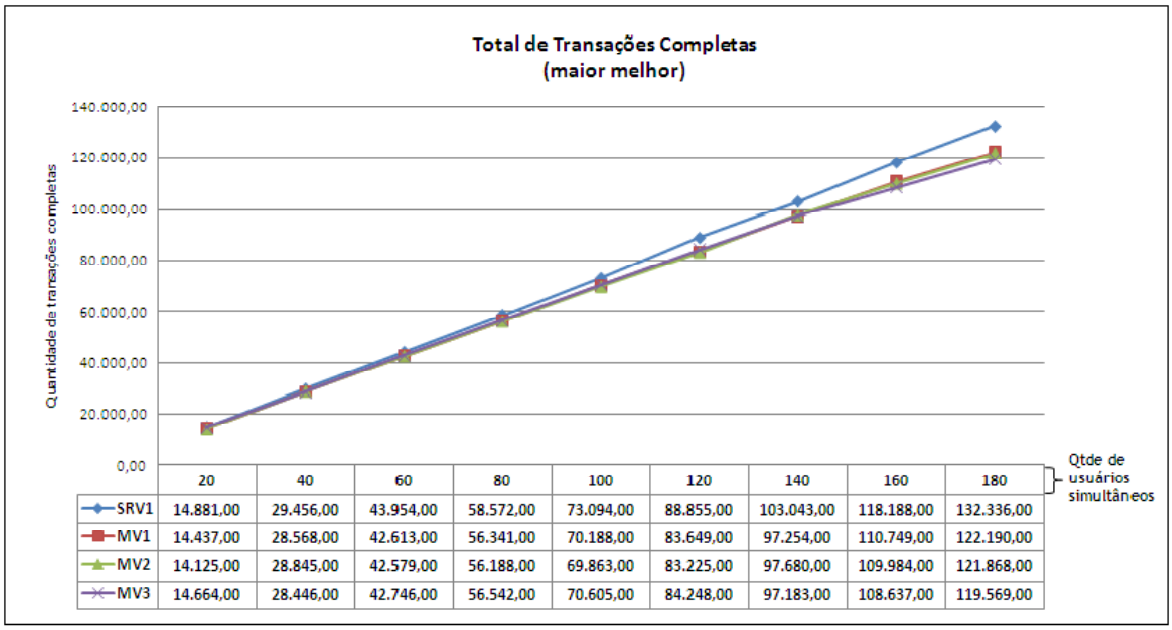
Recuperações e atualizações de dados em bancos de dados são feitas através de transações que são constituídas de uma ou mais sentenças *SQL (Structured Query Language)* para consulta a dados (*select*) e de um ou mais comandos (*insert, update, delete)* para alterações sobre dados. Os comandos comumente utilizados pelo *Swingbench* são: *insert* para a inclusão de novos dados, *update* para atualização de dados existentes, *deletes* para exclusão de dados gravados, *commits* para confirmação da transação em andamento e *rollbacks* para cancelamento da transação em andamento decorrente de alguma inconsistência ou falha. A Tabela 4.5 mostra para cada um dos comandos a quantidade executada nos testes realizados. Pode-se constatar que não houve nenhuma falha ou inconsistência que provocasse o cancelamento da transação (coluna Total de *Rollbacks*), bem como não foi executada nenhuma operação de exclusão de dados (coluna *Total de Deletes*), o que confere aos testes feitos confiabilidade nas execuções realizadas e integralidade de condições para os comparativos feitos.

Tabela 4.5 – Quantidade de comandos executados no banco de dados

Servidor	Total de *Selects*	Total de *Inserts*	Total de *Updates*	Total de *Deletes*	Total de *Commits*	Total de *Rollbacks*
SRV1	2.287.568	1.257.530,00	799.247	0	962.257	0
MV1	2.163.323	1.191.586,00	756.087	0	910.579	0
MV2	2.158.091	1.186.283,00	754.105	0	907.423	0
MV3	2.150.323	1.182.577,00	751.998	0	904.605	0
Total geral	**8.759.305**	**4.817.976,00**	**3.061.437**	**0**	**3.684.864**	**0**

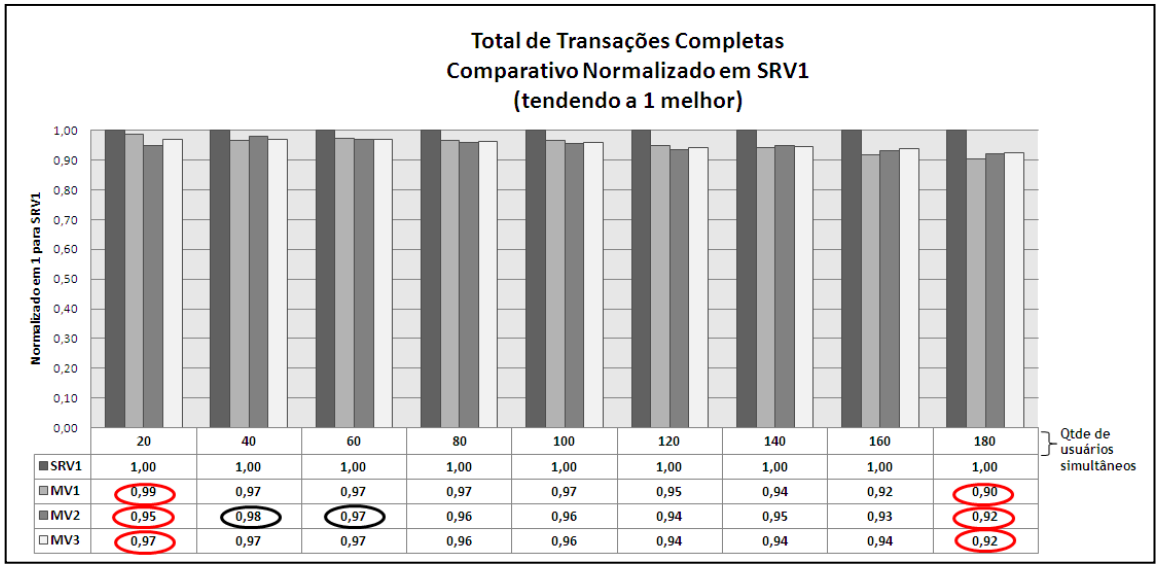
O Anexo F traz em maiores detalhes a sequência da execução desses comandos para as transações executadas.

A Figura 4.11 documenta o total de transações completas, isto é, finalizadas com sucesso, por cada servidor de banco de dados para os testes de 20, 40, 60, 80, 100, 120, 140, 160 e 180 usuários simultâneos, onde se constata que o total de transações completas no ambiente tradicional (SRV1) foi superior ao total executado nos ambientes virtualizados (MV1, MV2, MV3). Também se observa que à medida que se aumentava a quantidade de usuários simultâneos aumentava-se também a diferença entre o total de transações completadas pelo SRV1 e as completas pelas soluções com virtualização, apresentando o SRV1 uma maior capacidade de processamento de transações em banco de dados. A Figura 4.12 reflete essa diferença através da normalização da quantidade de transações em SRV1. Nesta normalização é possível verificar que inicialmente, com total de 20 usuários simultâneos, as quantidades de transações processadas pelas MV1, MV2 e MV3 foram bastante próximas do SRV1. A MV1 processou um total de transações correspondente a 99% do total processado pelo SRV1, a MV2 95% e a MV3 97%. Ao atingir os 180 usuários simultâneos, esses percentuais diminuem para 90%, 92% e 92% respectivamente. Esse comportamento indica uma maior capacidade do SRV1 em tratar e completar as transações visto que não possui a camada de abstração da virtualização.



	20	40	60	80	100	120	140	160	180
SRV1	14.881,00	29.456,00	43.954,00	58.572,00	73.094,00	88.855,00	103.043,00	118.188,00	132.336,00
MV1	14.437,00	28.568,00	42.613,00	56.341,00	70.188,00	83.649,00	97.254,00	110.749,00	122.190,00
MV2	14.125,00	28.845,00	42.579,00	56.188,00	69.863,00	83.225,00	97.680,00	109.984,00	121.868,00
MV3	14.664,00	28.446,00	42.746,00	56.542,00	70.605,00	84.248,00	97.183,00	108.637,00	119.569,00

Figura 4.11 – Total de transações completadas



	20	40	60	80	100	120	140	160	180
SRV1	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
MV1	0,99	0,97	0,97	0,97	0,97	0,95	0,94	0,92	0,90
MV2	0,95	0,98	0,97	0,96	0,96	0,94	0,95	0,93	0,92
MV3	0,97	0,97	0,97	0,96	0,96	0,94	0,94	0,94	0,92

Figura 4.12 – Total de transações completas – Normalizado em SRV1

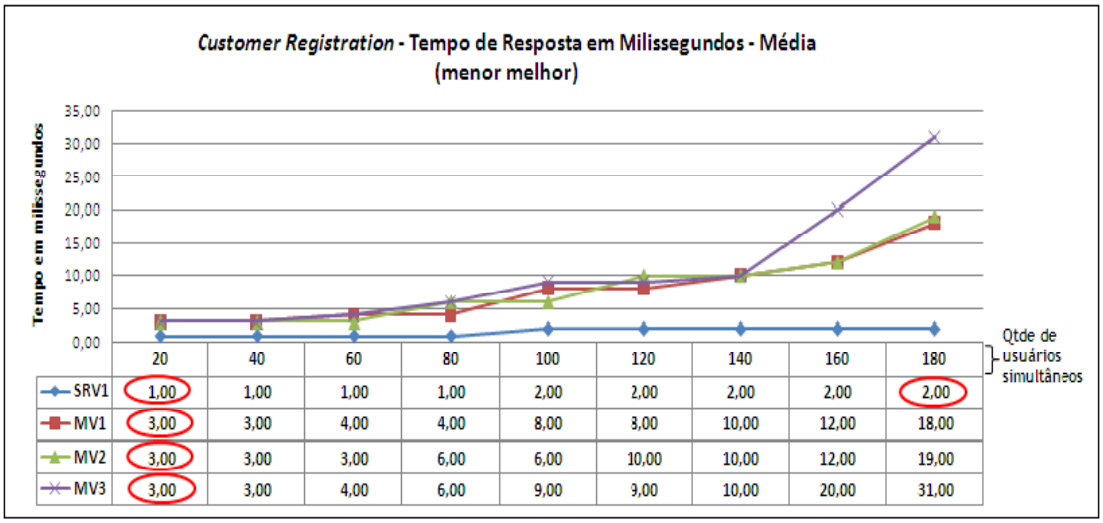
Esse resultado é compatível com resultados obtidos por testes de laboratórios feitos por fornecedores que também utilizaram a ferramenta *Swingbench.* Testes realizados pelo Grupo *Tolly*, por exemplo, avaliou a solução *Oracle VM* da *Oracle*, que é um MMV que utiliza a técnica paravirtualização (THE TOLLY GROUP, 2008). Os resultados obtidos identificaram uma eficiência do *Oracle VM* de 92,5% e 93,6% para 30 e 50 usuários simultâneos. Os resultados obtidos neste estudo para a solução que utiliza a paravirtualizaçao (MV2) atingiu de 98% a 97% para 40 e 60 usuários simultâneos. Embora escopos com quantidades de usuários simultâneos diferentes, eles são próximos e os resultados obtidos por este estudo foram mais eficientes.

4.2 - COMPARATIVOS DE TEMPO DE RESPOSTA DAS TRANSAÇÕES

Quanto aos tempos de resposta obtidos para as cinco transações do *benchmark Order Entry* - documentados através da Figura 4.13, Figura 4.15, Figura 4.17, Figura 4.19 e Figura 4.21 – constata-se que desde o primeiro teste (20 usuários simultâneos) os ambientes virtualizados (MV1, MV2 e MV3) gastaram mais tempo para completar as transações que o servidor tradicional (SRV1). Enquanto os tempos de respostas do SRV1 seguiram um perfil linear à medida que se aumentava a quantidade de usuários simultâneos, a MV1, MV2 e MV3 apresentaram perfil tendendo à semelhança de uma curva exponencial. Traçando um comparativo dessas cinco transações normalizadas em

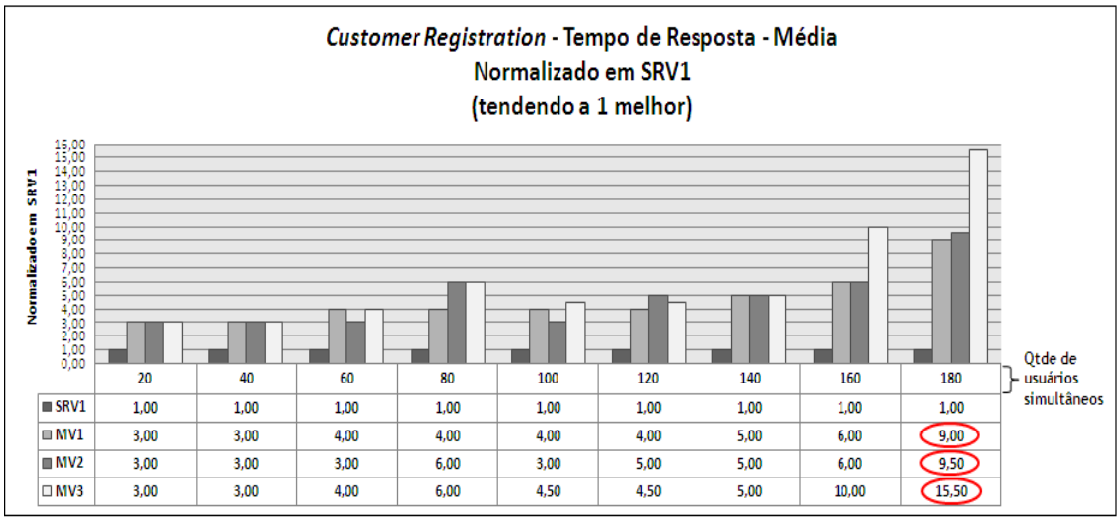
SRV1 conforme Figura 4.14, Figura 4.16, Figura 4.18, Figura 4.20 e Figura 4.22, foi possível apurar:

- A transação *Customer Registration*, conforme Figura 4.13, teve tempo médio de resposta entre um e dois milissegundos para o SRV1, tendo sido alcançado dois milissegundos a partir de 100 usuários simultâneos. As MV1, MV2 e MV3, já no primeiro teste, 20 usuários simultâneos, apresentaram tempo de resposta de três milissegundos, superior em 50% ao pior tempo registrado para o SRV1 (dois milissegundos) e equivalente a três vezes mais o melhor tempo registrado para o SRV1 (um milissegundo). Na Figura 4.14 pode-se observar que o tempo de resposta nas soluções com virtualização vai se degradando significativamente à medida que se aumenta a quantidade de usuários simultâneos. O pior desempenho foi atingido pela MV3, que no último teste precisou de 15,5 vezes mais tempo que o SRV1 para processar a mesma transação. Apesar da MV1 e MV2 não terem atingido esse tempo, ainda assim seus tempos no último teste foram relativamente altos se comparados ao SRV1, onde a MV1 precisou de 9 vezes mais tempo e a MV2 9,5 vezes mais.



	20	40	60	80	100	120	140	160	180
SRV1	1,00	1,00	1,00	1,00	2,00	2,00	2,00	2,00	2,00
MV1	3,00	3,00	4,00	4,00	8,00	8,00	10,00	12,00	18,00
MV2	3,00	3,00	3,00	6,00	6,00	10,00	10,00	12,00	19,00
MV3	3,00	3,00	4,00	6,00	9,00	9,00	10,00	20,00	31,00

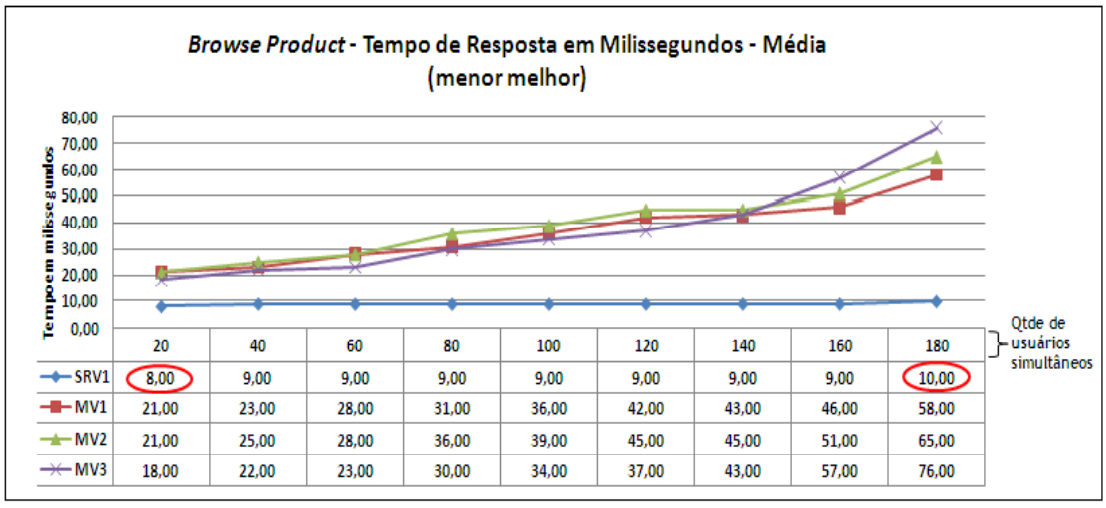
Figura 4.13 – *Customer Registration* – Tempo de resposta



	20	40	60	80	100	120	140	160	180
SRV1	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
MV1	3,00	3,00	4,00	4,00	4,00	4,00	5,00	6,00	9,00
MV2	3,00	3,00	3,00	6,00	3,00	5,00	5,00	6,00	9,50
MV3	3,00	3,00	4,00	6,00	4,50	4,50	5,00	10,00	15,50

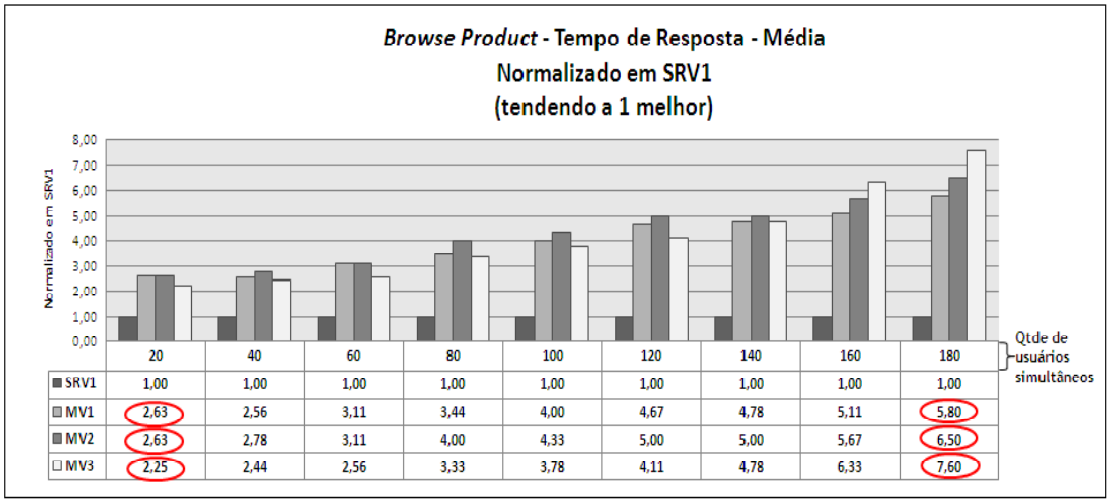
Figura 4.14 – *Customer Registration* – Tempo de resposta – Normalizado em SRV1

- A transação *Browser Product*, Figura 4.15, teve tempo médio de execução entre oito e dez milissegundos para o SRV1, tendo sido alcançado dez milissegundos somente no último teste (com 180 usuários simultâneos). A MV1, MV2 e MV3, já com 20 usuários simultâneos, e conforme Figura 4.16, gastaram, respectivamente, 2,63, 2,63 e 2,25 mais tempo que o SRV1. Novamente a MV3 apresentou o pior desempenho precisando, no último teste, de 7,6 vezes mais tempo que o SRV1, enquanto a MV1 e MV2 precisaram de 5,8 e 6,5 vezes mais, respectivamente.



	20	40	60	80	100	120	140	160	180
SRV1	8,00	9,00	9,00	9,00	9,00	9,00	9,00	9,00	10,00
MV1	21,00	23,00	28,00	31,00	36,00	42,00	43,00	46,00	58,00
MV2	21,00	25,00	28,00	36,00	39,00	45,00	45,00	51,00	65,00
MV3	18,00	22,00	23,00	30,00	34,00	37,00	43,00	57,00	76,00

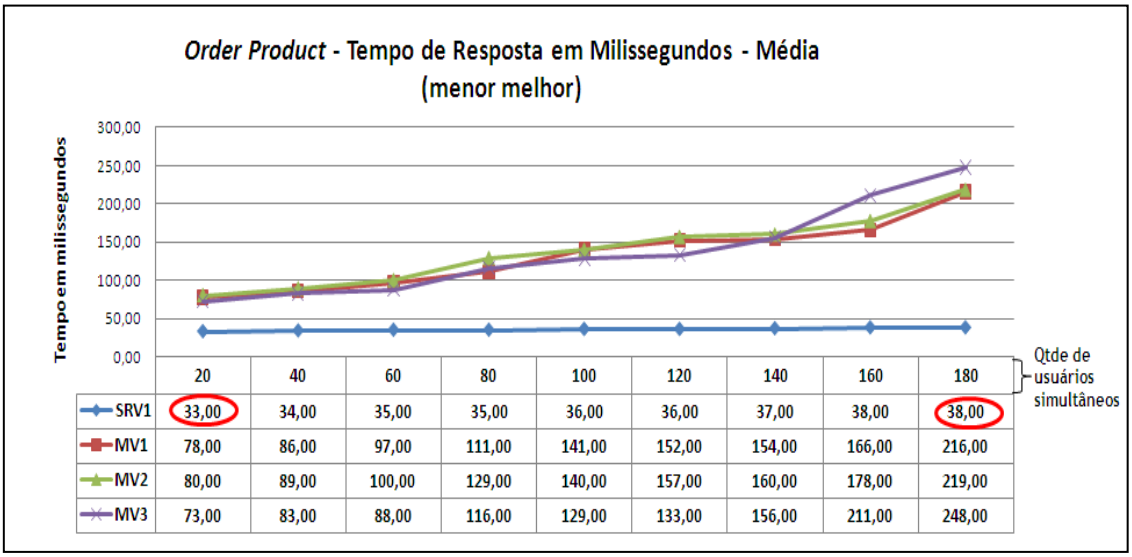
Figura 4.15 – *Browse Product* – Tempo de resposta



	20	40	60	80	100	120	140	160	180
SRV1	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
MV1	2,63	2,56	3,11	3,44	4,00	4,67	4,78	5,11	5,80
MV2	2,63	2,78	3,11	4,00	4,33	5,00	5,00	5,67	6,50
MV3	2,25	2,44	2,56	3,33	3,78	4,11	4,78	6,33	7,60

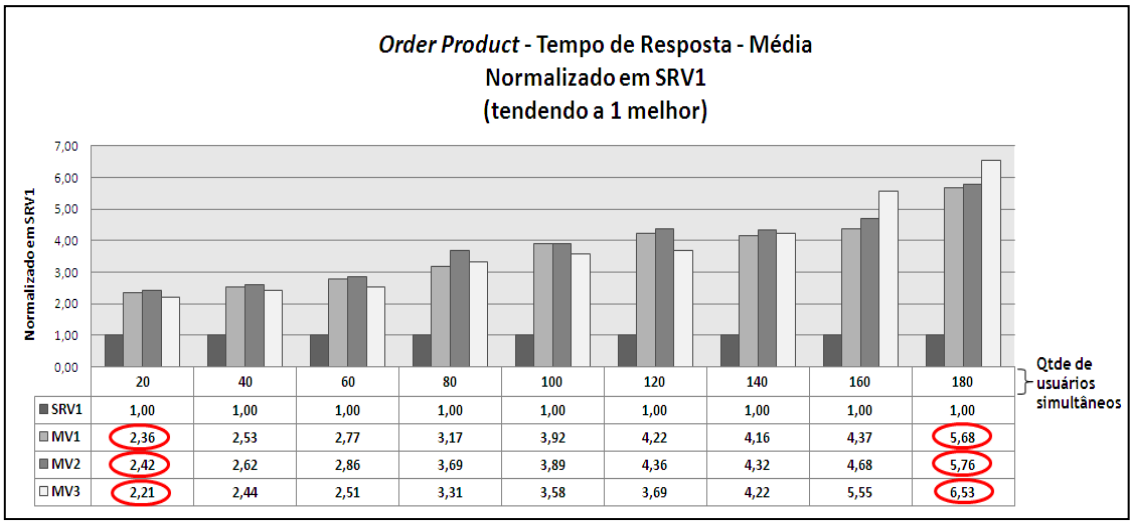
Figura 4.16 – *Browse Product* – Tempo de resposta – Normalizado em SRV1

- A transação *Order Product*, Figura 4.17, dentre as cinco transações testadas foi a que exigiu maior tempo de processamento. Para o SRV1 esse tempo variou de trinta e três a trinta e oito milissegundos à medida que a quantidade de usuários simultâneos aumentou. A MV1, MV2 e MV3 com 20 usuários simultâneos, e conforme Figura 4.18, gastaram, respectivamente, 2,36, 2,42 e 2,21 mais tempo que o SRV1. Seguindo o padrão apurado para as transações anteriores, aqui a MV3 também apresentou o pior desempenho precisando de 6,53 vezes mais tempo que o SRV1 para processar a transação no último teste, enquanto a MV1 e MV2 precisaram de 5,68 e 5,76 vezes mais, respectivamente.



	20	40	60	80	100	120	140	160	180
SRV1	33,00	34,00	35,00	35,00	36,00	36,00	37,00	38,00	38,00
MV1	78,00	86,00	97,00	111,00	141,00	152,00	154,00	166,00	216,00
MV2	80,00	89,00	100,00	129,00	140,00	157,00	160,00	178,00	219,00
MV3	73,00	83,00	88,00	116,00	129,00	133,00	156,00	211,00	248,00

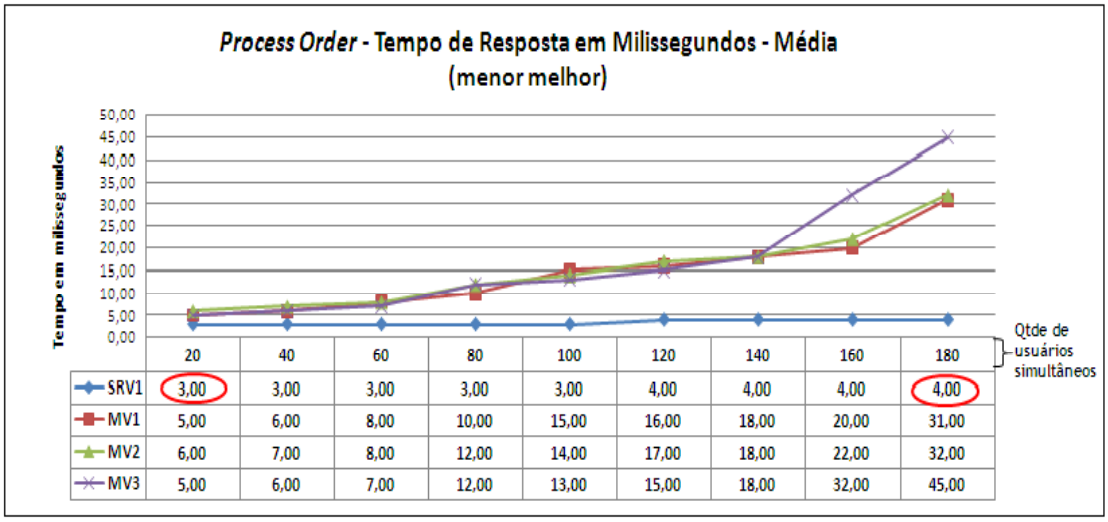
Figura 4.17 – *Order Product* – Tempo de resposta



	20	40	60	80	100	120	140	160	180
SRV1	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
MV1	2,36	2,53	2,77	3,17	3,92	4,22	4,16	4,37	5,68
MV2	2,42	2,62	2,86	3,69	3,89	4,36	4,32	4,68	5,76
MV3	2,21	2,44	2,51	3,31	3,58	3,69	4,22	5,55	6,53

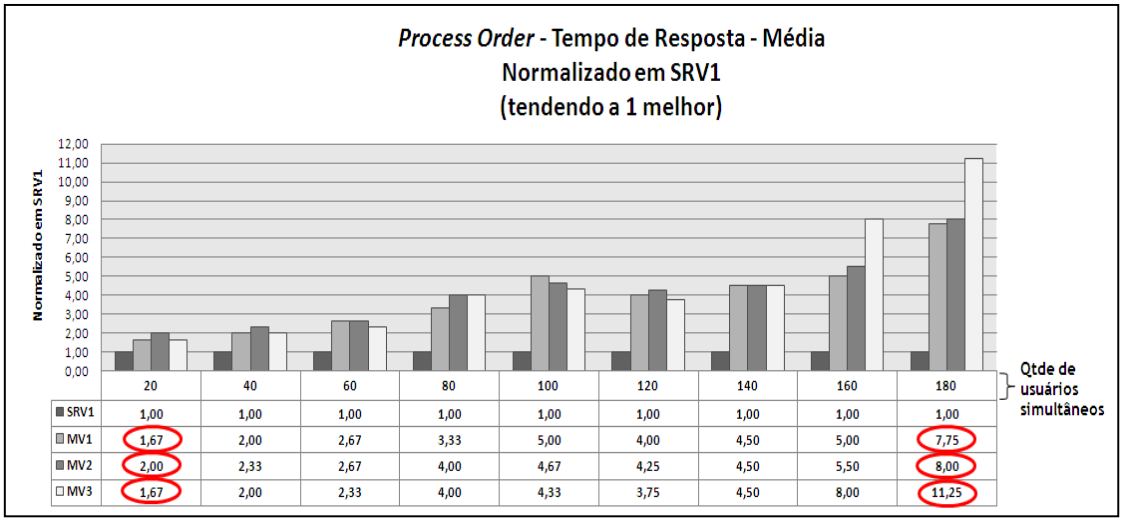
Figura 4.18 – *Order Product* – Tempo de resposta – Normalizado em SRV1

- A transação *Process Order*, Figura 4.19, apresentou para o SRV1 um tempo de resposta entre três e quatro milissegundos à medida que a quantidade de usuários simultâneos aumentava. A MV1, MV2 e MV3, também no teste inicial, e conforme Figura 4.20, gastaram, respectivamente, 1,67, 2 e 1,67 vezes mais tempo que o SRV1. Novamente a MV3 teve o pior desempenho precisando de 11,25 vezes mais tempo que o SRV1 para processar a transação no último teste, enquanto a MV1 e MV2 precisaram de 7,75 e 8 vezes mais respectivamente.



	20	40	60	80	100	120	140	160	180
SRV1	3,00	3,00	3,00	3,00	3,00	4,00	4,00	4,00	4,00
MV1	5,00	6,00	8,00	10,00	15,00	16,00	18,00	20,00	31,00
MV2	6,00	7,00	8,00	12,00	14,00	17,00	18,00	22,00	32,00
MV3	5,00	6,00	7,00	12,00	13,00	15,00	18,00	32,00	45,00

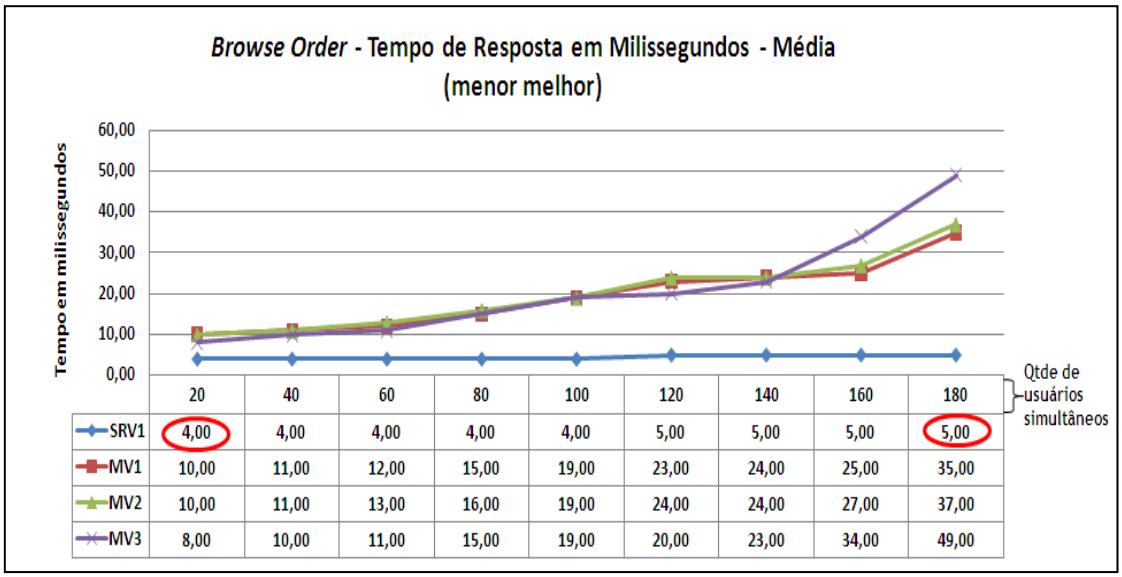
Figura 4.19 – *Process Order* – Tempo de resposta



	20	40	60	80	100	120	140	160	180
SRV1	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
MV1	1,67	2,00	2,67	3,33	5,00	4,00	4,50	5,00	7,75
MV2	2,00	2,33	2,67	4,00	4,67	4,25	4,50	5,50	8,00
MV3	1,67	2,00	2,33	4,00	4,33	3,75	4,50	8,00	11,25

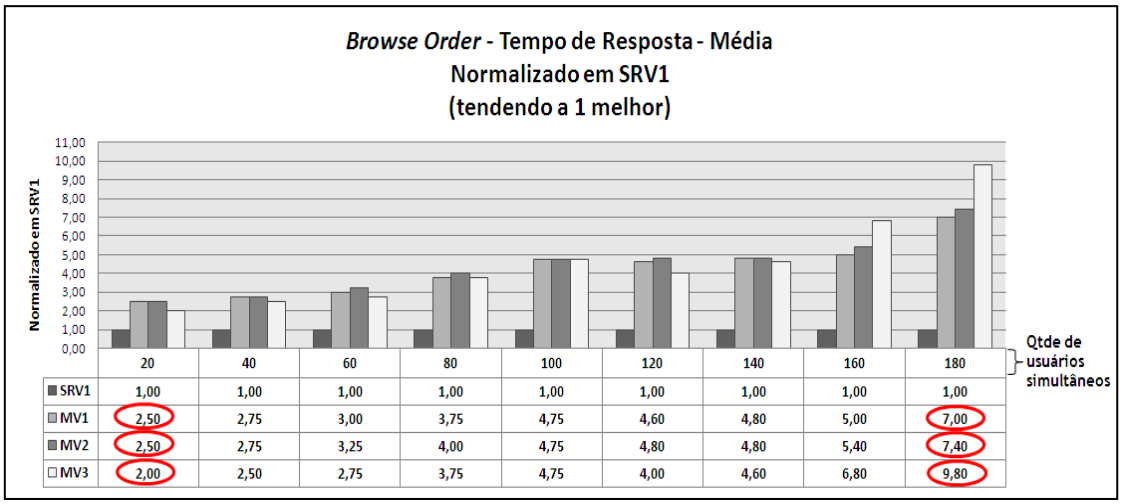
Figura 4.20 – *Process Order* – Tempo de resposta – Normalizado em SRV1

- Por fim, a transação *Browse Order*, Figura 4.21, foi completada pelo SRV1 com tempo de resposta entre quatro e cinco milissegundos à medida que a quantidade de usuários simultâneos aumentava. A MV1, MV2 e MV3, já no primeiro teste, e conforme Figura 4.22, gastaram, respectivamente, 2,5, 2,5 e 2 vezes mais tempo que o SRV1. Também nessa transação a MV3 obteve o pior desempenho precisando de 9,8 vezes mais tempo que o SRV1 no último teste, enquanto a MV1 e MV2 precisaram de 7 e 7,4 vezes mais respectivamente.



	20	40	60	80	100	120	140	160	180
SRV1	4,00	4,00	4,00	4,00	4,00	5,00	5,00	5,00	5,00
MV1	10,00	11,00	12,00	15,00	19,00	23,00	24,00	25,00	35,00
MV2	10,00	11,00	13,00	16,00	19,00	24,00	24,00	27,00	37,00
MV3	8,00	10,00	11,00	15,00	19,00	20,00	23,00	34,00	49,00

Figura 4.21 – *Browse Order* – Tempo de resposta



	20	40	60	80	100	120	140	160	180
SRV1	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
MV1	2,50	2,75	3,00	3,75	4,75	4,60	4,80	5,00	7,00
MV2	2,50	2,75	3,25	4,00	4,75	4,80	4,80	5,40	7,40
MV3	2,00	2,50	2,75	3,75	4,75	4,00	4,60	6,80	9,80

Figura 4.22 – *Browse Order* – Tempo de resposta – Normalizado em SRV1

Com os resultados apurados, fica evidenciado que o tempo de resposta médio para as cinco transações do *Order Entry* apresentou aumento significativo para as soluções com virtualização (MV1, MV2 e MV3) se comparadas com o ambiente tradicional (SRV1), conforme resumo documentado na Tabela 4.6. O aumento mínimo registrado ocorreu na transação *Process Order*, onde o SRV1 teve tempo de resposta de três milissegundos e a MV1 cinco milissegundos, uma diferença de 166,67%. O aumento máximo apurado foi de 1.550% na transação *Customer Registration*, com o SRV1 respondendo em dois milissegundos e a MV3 em trinta e um milissegundos.

Tabela 4.6 – Quadro resumo dos tempos de resposta das transações

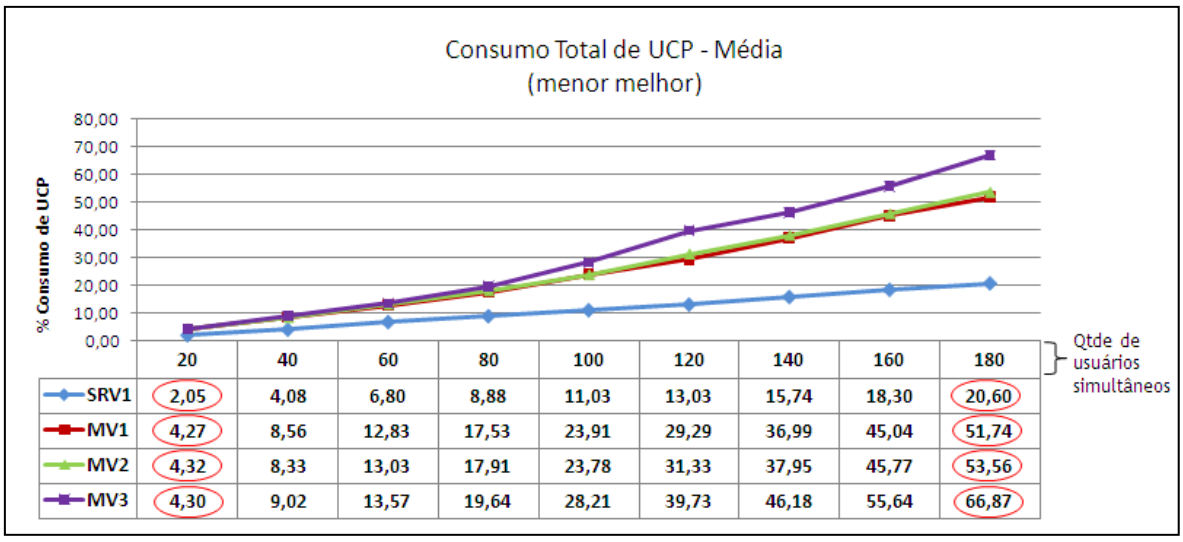
Transação	Servidor	Tempo de Resposta (em milissegundos)		Comparativo Normalizado em SRV1	
		Mínimo	Máximo	Mínimo	Máximo
Customer Registration	**SRV1**	**1,00**	**2,00**		
	MV1	3,00	18,00	300,00%	900,00%
	MV2	3,00	19,00	300,00%	950,00%
	MV3	3,00	31,00	300,00%	1550,00%
Browse Product	**SRV1**	**8,00**	**10,00**		
	MV1	21,00	58,00	262,50%	580,00%
	MV2	21,00	65,00	262,50%	650,00%
	MV3	18,00	76,00	225,00%	760,00%
Order Product	**SRV1**	**33,00**	**38,00**		
	MV1	78,00	216,00	236,36%	568,42%
	MV2	80,00	219,00	242,42%	576,32%
	MV3	73,00	248,00	221,21%	652,63%
Process Order	**SRV1**	**3,00**	**4,00**		
	MV1	5,00	31,00	166,67%	775,00%
	MV2	6,00	32,00	200,00%	800,00%
	MV3	5,00	45,00	166,67%	1125,00%
Browse Order	**SRV1**	**4,00**	**5,00**		
	MV1	10,00	35,00	250,00%	700,00%
	MV2	10,00	37,00	250,00%	740,00%
	MV3	8,00	49,00	200,00%	980,00%

Diante desses resultados, concluí-se que a camada de virtualização, à medida que se aumenta a carga (neste estudo provocado pelo aumento de usuários simultâneos), implica em degradação do tempo de resposta.

4.3 - COMPARATIVOS DE USO DE UCP

Ao analisar o consumo de UCP de um servidor, se espera que ele gaste mais tempo atendendo as solicitações dos aplicativos e usuários do que resolvendo chamadas de sistema. Situação em que há pouco uso de UCP em modo *kernel* significa maior disponibilidade da UCP para atender as solicitações da aplicação e usuários e, por conseguinte, menor janela de processamento ou menor tempo de resposta. Esse é, tipicamente, o comportamento no mundo tradicional, onde ocorrências de um uso intensivo em modo *kernel*, comumente, indicam alguma contenção de *hardware.*

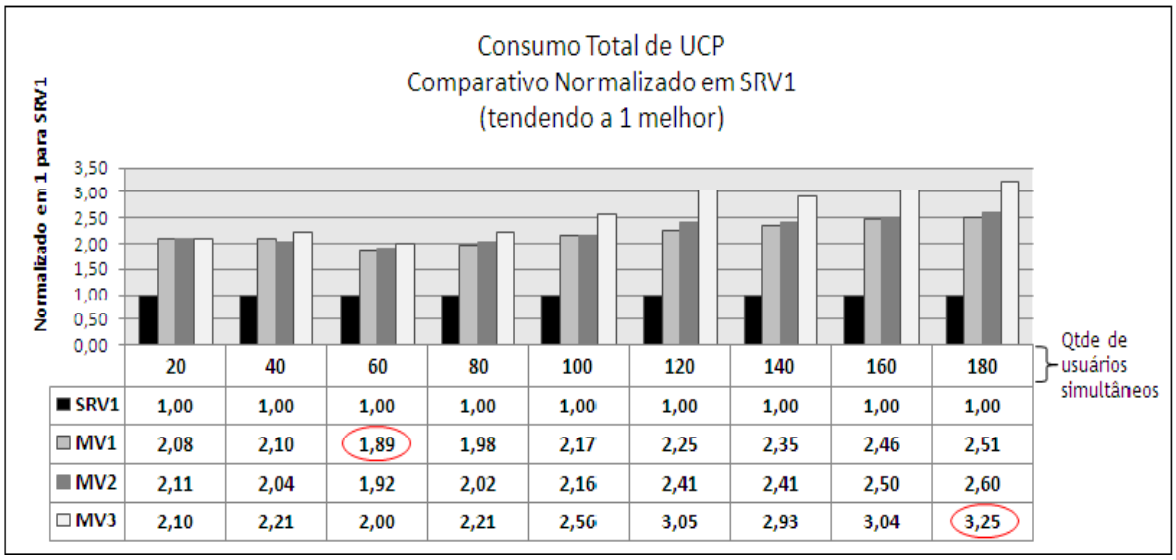
A análise do consumo total médio de UCP, conforme detalhado na Figura 4.23, aponta que o consumo do SRV1 se manteve em patamares inferiores às soluções com virtualização (MV1, MV2, MV3). Enquanto o SRV1 utilizou média de 2,05% de UCP para tratar 20 usuários simultâneos, chegando a 20,6% ao tratar 180, a MV1, MV2 e MV3 iniciaram em 4,27%, 4,32% e 4,3% respectivamente e atingiram, no final, 51,74%, 53,56% e 66,87%, também respectivamente.



	20	40	60	80	100	120	140	160	180
SRV1	2,05	4,08	6,80	8,88	11,03	13,03	15,74	18,30	20,60
MV1	4,27	8,56	12,83	17,53	23,91	29,29	36,99	45,04	51,74
MV2	4,32	8,33	13,03	17,91	23,78	31,33	37,95	45,77	53,56
MV3	4,30	9,02	13,57	19,64	28,21	39,73	46,18	55,64	66,87

Figura 4.23 – Consumo total de UCP – Média

Essa diferença de uso de consumo de UCP entre o SRV1 e a MV1, MV2 e MV3 está normalizada na Figura 4.24, onde se apura que a camada de virtualização exige mais consumo de UCP, em proporções que variam entre 1,89 e 3,25 vezes.



	20	40	60	80	100	120	140	160	180
SRV1	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
MV1	2,08	2,10	1,89	1,98	2,17	2,25	2,35	2,46	2,51
MV2	2,11	2,04	1,92	2,02	2,16	2,41	2,41	2,50	2,60
MV3	2,10	2,21	2,00	2,21	2,56	3,05	2,93	3,04	3,25

Figura 4.24 – Consumo total de UCP – Média – Normalizado em SRV1

Desmembrando-se o consumo total de UCP em modo usuário e modo *kernel*, Figura 4.25, constata-se um comportamento similar no uso de UCP em modo usuário para todos os servidores avaliados, apresentando os ambientes virtualizados um consumo um pouco maior. Consumo em modo usuário reflete disponibilidade do servidor em tratar as necessidades específicas da aplicação, o que é bom. Por outro lado, observa-se um consumo expressivamente maior em modo *kernel* para os ambientes virtualizados. Enquanto o SRV1 utilizou média de 0,03% para tratar 20 usuários simultâneos, chegando a 2,03% ao tratar 180, a MV1, MV2 e MV3 já na segunda bateria de testes (40 usuários simultâneos) usaram mais tempo de UCP em modo *kernel* que o SRV1 ao tratar 180 usuários simultâneos (3,35%, 3,35% e 3,73% respectivamente).

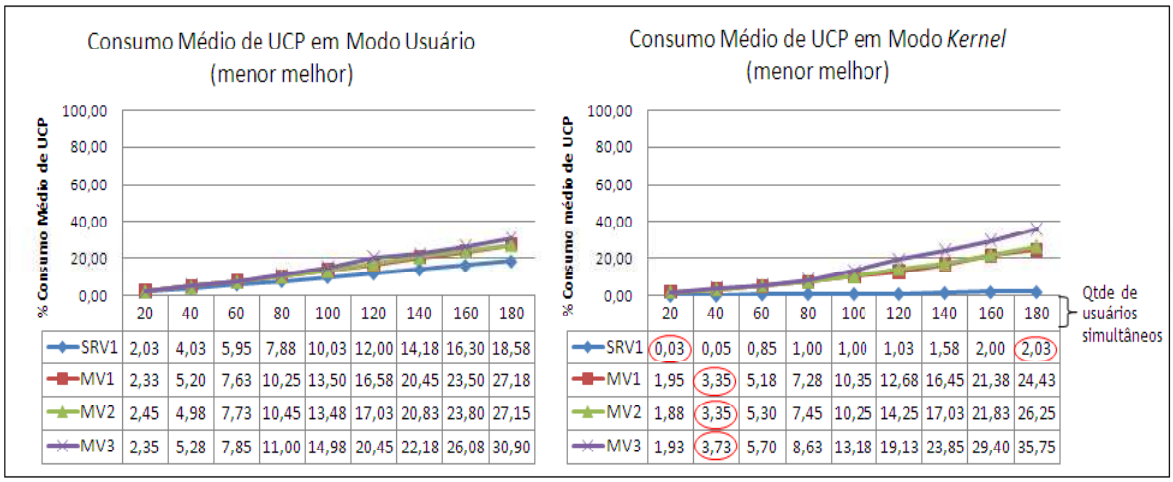


Figura 4.25 – Consumo médio de UCP em modo usuário e *kernel*

Calculando-se o consumo proporcional entre modo usuário e *kernel* - resultados documentados na Figura 4.26 - confirma-se que os ambientes virtualizados exigem mais consumo de UCP em modo *kernel* que o servidor tradicional. Enquanto o SRV1 consumiu entre 1% e 13% da UCP em modo *kernel*, a MV1 usou de 39% a 48%, a MV2 de 40% a 49% e a MV3 entre 41% e 54%.

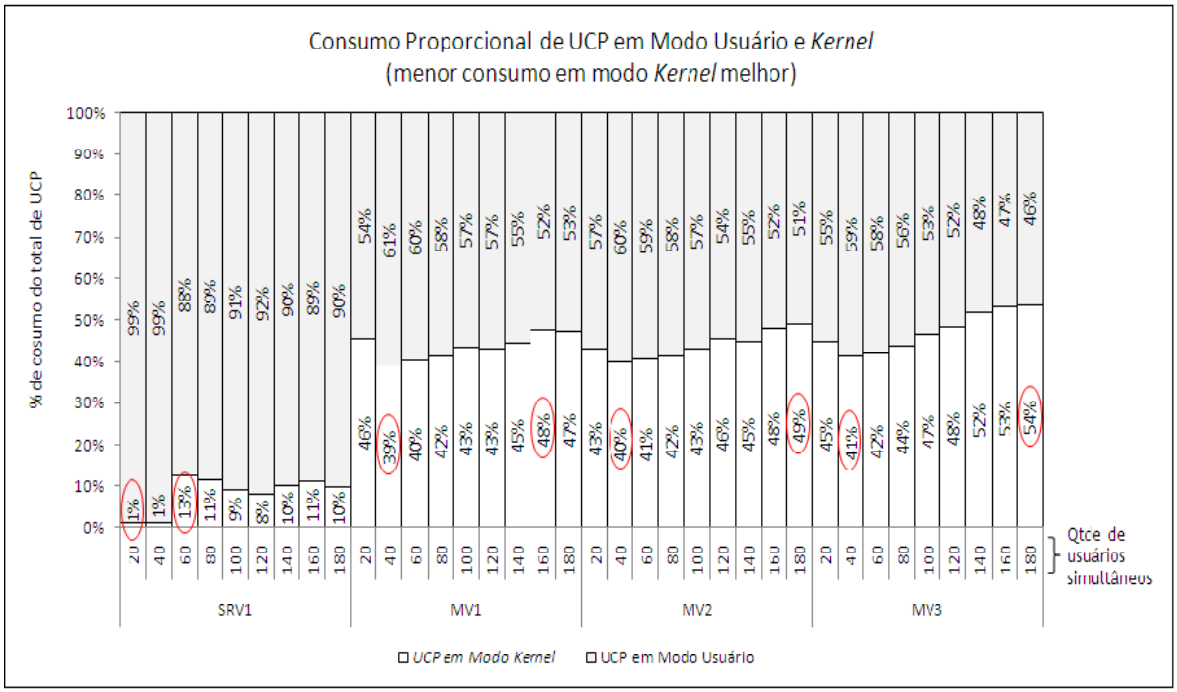


Figura 4.26 – Consumo proporcional de UCP em modo usuário e modo *Kernel*

Com os resultados apurados é possível concluir que nos ambientes virtualizados há uma tendência maior de chamadas de sistema ao SO, o que contribui por uma degradação no quesito tempo de resposta.

4.4 - COMPARATIVOS DE USO DE MEMÓRIA

Não foi registrada nenhuma anomalia no uso de memória decorrente de uma variação brusca ou elevada, ou devido a sua insuficiência. A falta de memória comumente provoca a retirada de páginas da memória física (RAM) para disco (área de *swap*) para liberar memória para os processos ativos. Tanto podem ser migradas algumas páginas de um processo (mecanismo chamado de paginação – em inglês *paging*) ou todas as páginas de um processo (chamado de *swapping* – sem tradução para o português). Uma vez migrada uma página para a área de *swap*, quando o processo dono dessa página a referenciar novamente, será necessário retorná-la para a memória física, A Tabela 4.7 mostra que não houve ocorrências de retirar ou retornar páginas da área de *swap*.

Tabela 4.7 – Estatísticas de troca de páginas entre a RAM e área de SWAP

Qtde usuários simultâneos	Número de páginas retiradas da RAM para a área de *swap por* segundo				Número de páginas retornadas da área de *swap* para a RAM por segundo			
	SRV1	MV1	MV2	MV3	SRV1	MV1	MV2	MV3
20	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0
60	0	0	0	0	0	0	0	0
80	0	0	0	0	0	0	0	0
100	0	0	0	0	0	0	0	0
120	0	0	0	0	0	0	0	0
140	0	0	0	0	0	0	0	0
160	0	0	0	0	0	0	0	0
180	0	0	0	0	0	0	0	0
Total geral	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

Também não foi observado consumo significativo de memória em nenhum dos quatro servidores. Em linhas gerais, praticamente todos os servidores tiveram o mesmo consumo: em torno de 2,48GB de memória para atendimento aos aplicativos e usuários e 2,32GB para o *kernel*, sendo 0,02GB como área de *buffer* e 2,29GB como área de *cache.* As tabelas Tabela 4.8, Tabela 4.9, Tabela 4.10 e Tabela 4.11 retratam o consumo de memória de todos os servidores em todos os testes realizados, onde o dado apresentado em cada coluna representa:

- Quantidade de memória disponível: total de memória RAM livre.
- Quantidade de memória utilizada: total de memória RAM em uso.
- Quantidade de memória usada como *buffer*: total de memória RAM utilizada temporariamente para escrita e leitura de dados. Os dados podem ser originados de dispositivos externos ou internos ao sistema. São utilizados quando existe uma diferença entre a taxa em que os dados são recebidos e a taxa em que eles podem ser processados, ou no caso em que estas taxas são variáveis. Favorece liberar uma aplicação enquanto a escrita em disco, por exemplo, finaliza.
- Quantidade de memória usada como *cache*: total da memória RAM utilizada para manter os dados ou arquivos mais recentemente utilizados;
- Área de *swap* livre: total da área de *swap* livre;
- Área de *swap* utilizada: total da área de *swap* utilizada.

Tabela 4.8 – Estatísticas de consumo de memória SRV1 – Média por segundo

Valores em GB (Gigabytes)							
Qtde usuários simultâneos	Servidor	Quantidade de memória disponível	Quantidade de memória utilizada	Quantidade de memória usada como *buffer*	Quantidade de memória usada como *cache*	Área de swap livre	Área de swap utilizada
20	SRV1	26,84	2,48	0,02	2,29	12,20	0
40	SRV1	26,47	2,48	0,02	2,29	12,20	0
60	SRV1	26,15	2,48	0,02	2,29	12,20	0
80	SRV1	25,90	2,48	0,02	2,29	12,20	0
100	SRV1	25,70	2,48	0,02	2,29	12,20	0
120	SRV1	25,48	2,48	0,02	2,29	12,20	0
140	SRV1	25,27	2,48	0,02	2,29	12,20	0
160	SRV1	25,08	2,48	0,02	2,29	12,20	0
180	SRV1	24,88	2,48	0,02	2,29	12,20	0

Tabela 4.9 – Estatísticas de consumo de memória MV1 – Média por segundo

Valores em GB (Gigabytes)							
Qtde usuários simultâneos	Servidor	Quantidade de memória disponível	Quantidade de memória utilizada	Quantidade de memória usada como *buffer*	Quantidade de memória usada como *cache*	Área de swap livre	Área de swap utilizada
20	MV1	27,30	2,48	0,02	2,29	12,20	0
40	MV1	27,49	2,48	0,02	2,29	12,20	0
60	MV1	26,32	2,48	0,02	2,29	12,20	0
80	MV1	27,03	2,48	0,02	2,29	12,20	0
100	MV1	26,49	2,48	0,02	2,29	12,20	0
120	MV1	26,58	2,48	0,02	2,29	12,20	0
140	MV1	26,09	2,48	0,02	2,29	12,20	0
160	MV1	25,90	2,48	0,02	2,29	12,20	0
180	MV1	25,77	2,48	0,02	2,29	12,20	0

Tabela 4.10 – Estatísticas de consumo de memória MV2 – Média por segundo

Valores em GB (Gigabytes)							
Qtde usuários simultâneos	Servidor	Quantidade de memória disponível	Quantidade de memória utilizada	Quantidade de memória usada como *buffer*	Quantidade de memória usada como *cache*	Área de swap livre	Área de swap utilizada
20	MV2	27,60	2,48	0,02	2,29	12,20	0
40	MV2	27,24	2,48	0,02	2,29	12,20	0
60	MV2	26,95	2,48	0,02	2,29	12,20	0
80	MV2	26,74	2,48	0,02	2,29	12,20	0
100	MV2	26,56	2,48	0,02	2,29	12,20	0
120	MV2	26,40	2,48	0,02	2,29	12,20	0
140	MV2	26,26	2,48	0,02	2,29	12,20	0
160	MV2	26,14	2,48	0,02	2,29	12,20	0
180	MV2	26,02	2,48	0,02	2,29	12,20	0

Tabela 4.11 – Estatísticas de consumo de memória MV3 – Média por segundo

Valores em GB (Gigabytes)							
Qtde usuários simultâneos	Servidor	Quantidade de memória disponível	Quantidade de memória utilizada	Quantidade de memória usada como *buffer*	Quantidade de memória usada como *cache*	Área de swap livre	Área de swap utilizada
20	MV3	27,63	2,48	0,02	2,29	12,20	0
40	MV3	27,07	2,48	0,02	2,29	12,20	0
60	MV3	26,84	2,48	0,02	2,29	12,20	0
80	MV3	26,67	2,48	0,02	2,29	12,20	0
100	MV3	26,52	2,48	0,02	2,29	12,20	0
120	MV3	26,00	2,48	0,02	2,29	12,20	0
140	MV3	25,85	2,48	0,02	2,29	12,20	0
160	MV3	25,75	2,48	0,02	2,29	12,20	0
180	MV3	25,63	2,48	0,02	2,29	12,20	0

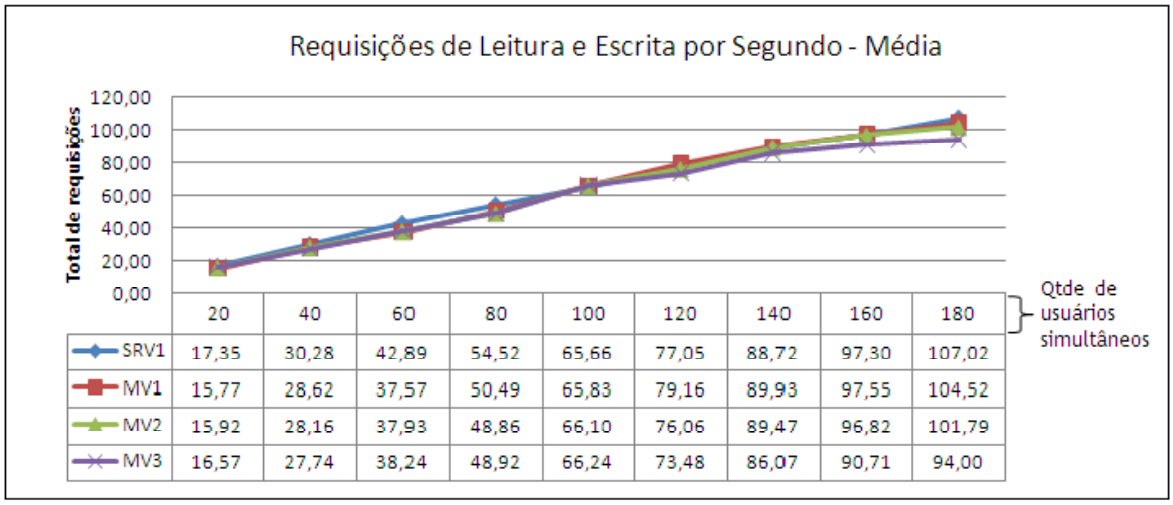
Diante dos resultados obtidos, no qual os servidores apresentaram valores praticamente iguais para todos os quesitos analisados (quantidade de memória disponível, quantidade de memória utilizada, quantidade de memória usada como *buffer*, quantidade de memória usada como *cache*, área de *swap* livre e área de *swap* utilizada), conclui-se que o uso da memória em ambiente virtualizado mantém-se em patamares similares ao tradicional.

4.5 - COMPARATIVOS DE ACESSO A DISCO (LEITURA E ESCRITA)

A análise de desempenho a disco requer avaliar a disponibilidade de área em disco e investigar a vazão do atendimento conforme a quantidade de requisições de acessos de leitura e escrita solicitada pela aplicação.

Em relação à disponibilidade, a área em disco utilizada para os testes foi previamente configurada para suportar toda a área do banco de dados de 3,7GB no diretório */oracle* que foi criado com 9.7GB, portanto seu uso foi de aproximadamente 40%, com 60% de área livre (detalhes no Anexo E).

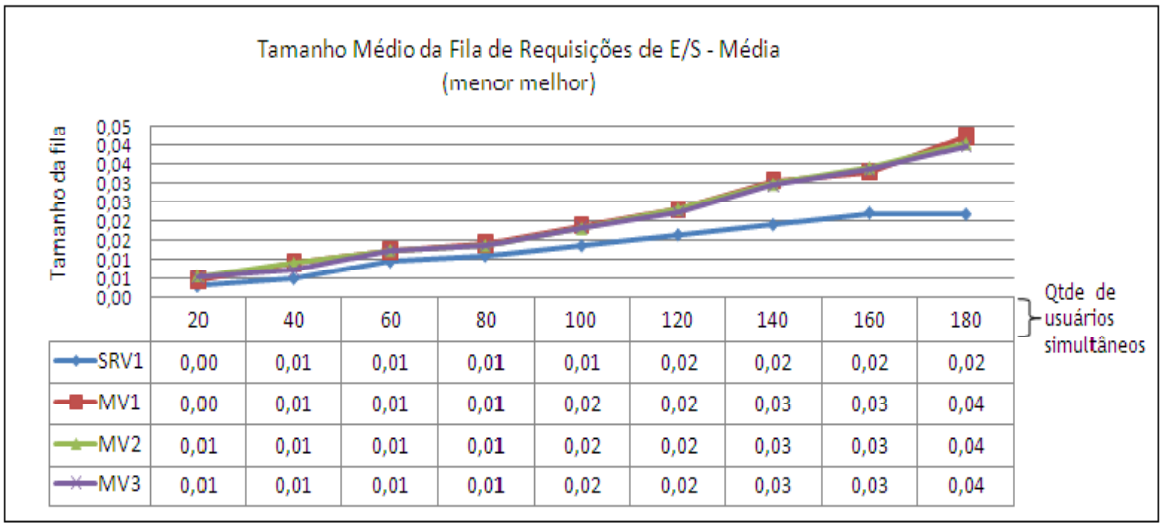
Quanto à capacidade do servidor em atender as requisições feitas, avaliemos inicialmente o total médio de requisições de leitura e escrita por segundo comandadas pelo aplicativo e documentado na Figura 4.27. Observa-se que o total de requisições foi bastante similar entre os servidores, havendo pouca variação entre o total de operações apurado para o SRV1 e a MV1, MV2 e MV3.



	20	40	60	80	100	120	140	160	180
SRV1	17,35	30,28	42,89	54,52	65,66	77,05	88,72	97,30	107,02
MV1	15,77	28,62	37,57	50,49	65,83	79,16	89,93	97,55	104,52
MV2	15,92	28,16	37,93	48,86	66,10	76,06	89,47	96,82	101,79
MV3	16,57	27,74	38,24	48,92	66,24	73,48	86,07	90,71	94,00

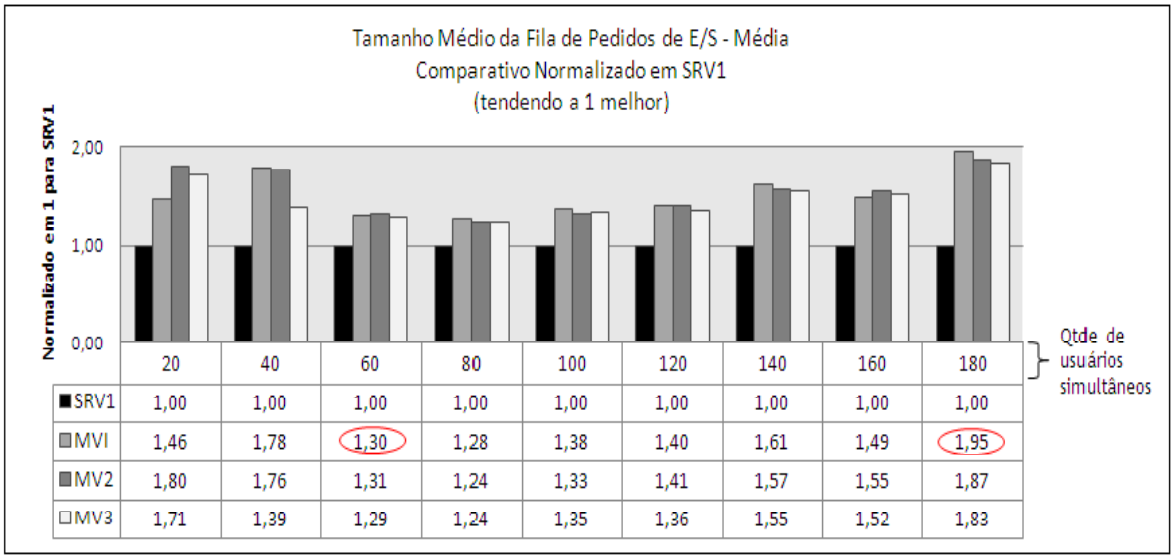
Figura 4.27 – Requisições de leitura e escrita por segundo – Média

O tamanho médio da fila de requisições de leitura e escrita, conforme exposto na Figura 4.28, aponta uma média abaixo de uma requisição na fila para todos os servidores. Entretanto, ao se comparar a MV1, MV2 e MV3 com o SRV1, Figura 4.29, o tamanho médio da fila do MV1, MV2 e MV3 é sempre maior, oscilando entre 1,3 e 1,95 a mais de requisições na fila.



	20	40	60	80	100	120	140	160	180
SRV1	0,00	0,01	0,01	0,01	0,01	0,02	0,02	0,02	0,02
MV1	0,00	0,01	0,01	0,01	0,02	0,02	0,03	0,03	0,04
MV2	0,01	0,01	0,01	0,01	0,02	0,02	0,03	0,03	0,04
MV3	0,01	0,01	0,01	0,01	0,02	0,02	0,03	0,03	0,04

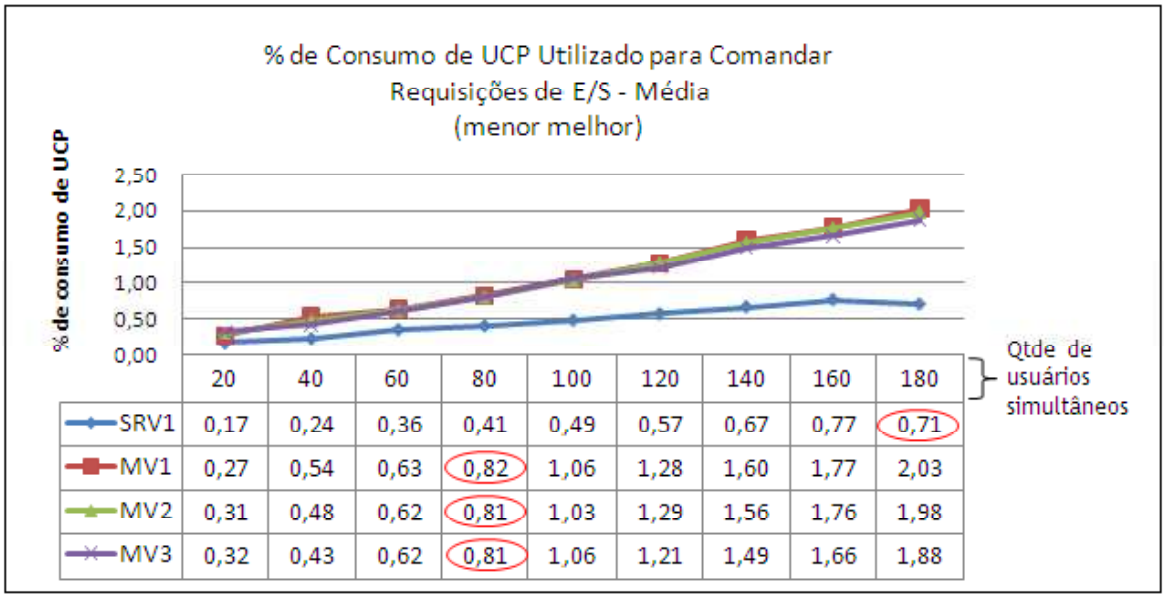
Figura 4.28 – Tamanho da fila de requisições de E/S – Média



	20	40	60	80	100	120	140	160	180
SRV1	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
MVI	1,46	1,78	1,30	1,28	1,38	1,40	1,61	1,49	1,95
MV2	1,80	1,76	1,31	1,24	1,33	1,41	1,57	1,55	1,87
MV3	1,71	1,39	1,29	1,24	1,35	1,36	1,55	1,52	1,83

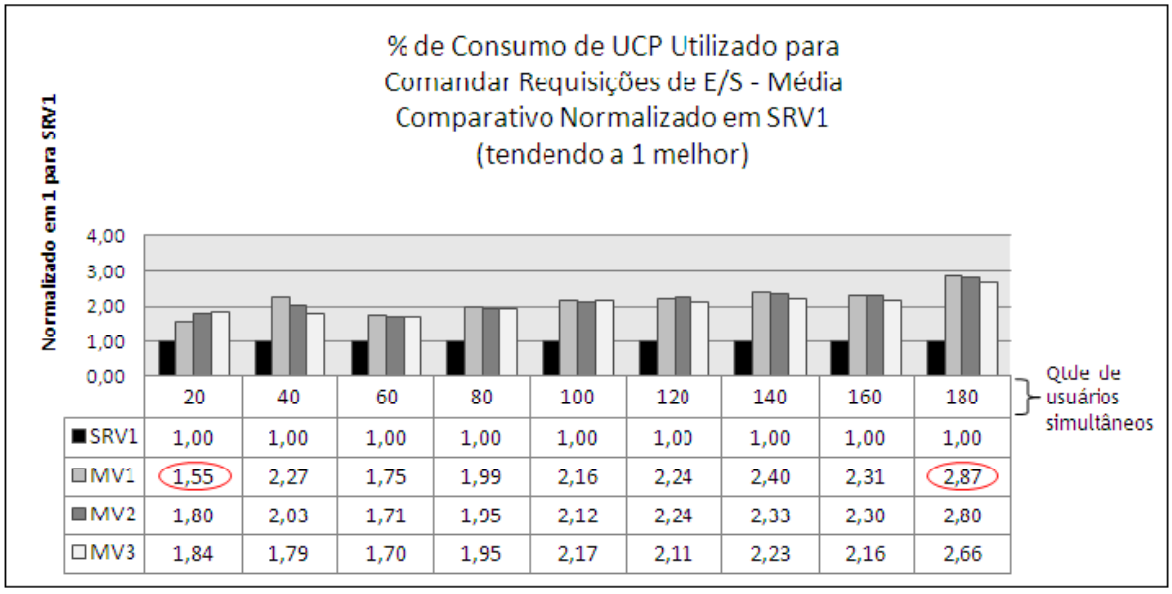
Figura 4.29 – Tamanho da fila de requisições de E/S – Média – Normalizado em SRV1

Conforme Figura 4.30, o tempo de UCP gasto para comandar os pedidos de leitura e escrita para o SRV1 não ultrapassou 0,71%, no teste de maior quantidade de usuários simultâneos (180), enquanto que a MV1, MV2 e MV3 superou este percentual já a partir dos 80 usuários simultâneos. Normalizando esse consumo de UCP em SRV1 (Figura 4.31), constata-se que a MV1, MV2 e MV3 gastaram entre 1,55 e 2,87 vezes mais UCP que o SRV1 gastou para comandar os pedidos de leitura e escrita para disco.



	20	40	60	80	100	120	140	160	180
SRV1	0,17	0,24	0,36	0,41	0,49	0,57	0,67	0,77	0,71
MV1	0,27	0,54	0,63	0,82	1,06	1,28	1,60	1,77	2,03
MV2	0,31	0,48	0,62	0,81	1,03	1,29	1,56	1,76	1,98
MV3	0,32	0,43	0,62	0,81	1,06	1,21	1,49	1,66	1,88

Figura 4.30 – % UCP para comandar requisições de E/S – Média



	20	40	60	80	100	120	140	160	180
SRV1	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
MV1	1,55	2,27	1,75	1,99	2,16	2,24	2,40	2,31	2,87
MV2	1,80	2,03	1,71	1,95	2,12	2,24	2,33	2,30	2,80
MV3	1,84	1,79	1,70	1,95	2,17	2,11	2,23	2,16	2,66

Figura 4.31 – % UCP para comandar requisições de E/S – Média – Normalizado em SRV1

Compilando-se os resultados obtidos, constata-se similaridade no total médio de requisições de leitura e escrita em disco, onde o tamanho da fila apresentou média abaixo de uma requisição para todos os servidores. Entretanto, foi observado que o tempo gasto pela UCP para emitir o pedido de leitura e escrita foi maior para as soluções com virtualização, que gastaram entre 1,55 e 2,87 vezes mais tempo que o SRV1. Esse resultado indica que os ambientes virtualizados apresentam maior latência de E/S, o que também contribui por uma degradação no quesito tempo de resposta.

4.6 - COMPARATIVOS DE ATENDIMENTO A PROCESSOS E SEGMENTOS

A vazão de um servidor para tratar uma carga imposta pode ser percebida pela quantidade de processos que ficam em fila à espera de UCP e pelo total de processos e segmentos constantes na lista de processos. Uma maior fila à espera de UCP indica maior tempo para atendimento às requisições feitas pelas aplicações. Uma maior lista de processos indica uma maior carga de processamento sendo imposta ao servidor em decorrência das requisições emitidas pelas aplicações.

A Tabela 4.12 apresenta os resultados apurados em todos os testes realizados para o total de processos aguardando por UCP e o total de processos e segmentos constantes na listas de processos, e os normaliza em SRV1.

Analisando os resultados do número de processos aguardando por UCP, atesta-se que o SRV1 apresentou uma menor fila em quase todos os testes realizados, ocorrendo uma única exceção no teste com 20 usuários, onde a MV3 manteve, na média, 29% menos de processos na fila ao se comparar com o SRV1. Para os demais testes, a menor fila observada para o ambiente virtualizado foi obtida no teste de 60 usuários simultâneos onde a MV2 teve, na média, 13% mais de processos na fila que o SRV1. A maior fila ocorreu no teste de 180 usuários, para a MV3, onde houve 718% mais processos aguardando na fila ao se comparar com o SRV1.

Em relação ao total de processos e segmentos constantes na lista de processos, em todos os testes realizados os ambientes virtualizados apresentaram uma menor quantidade se comparados ao SRV1. A menor diferença ocorreu no teste de 60 usuários, onde a lista do SRV1 estava com 278 processos e segmentos e a MV1 com 269, ou seja, uma diferença a menor para a MV1 de 3,42%. A maior diferença também ocorreu entre o SRV1 e a MV1, teste de 120 usuários simultâneos, tendo a fila do SRV 350 processos e segmentos e a MV1 316, uma diferença percentual a menor para a MV1 de 9,60%.

Os resultados apurados indicam que nos ambientes virtualizados há um maior enfileiramento de processos e segmentos, independente da quantidade de processos e segmentos constantes na lista de processos.

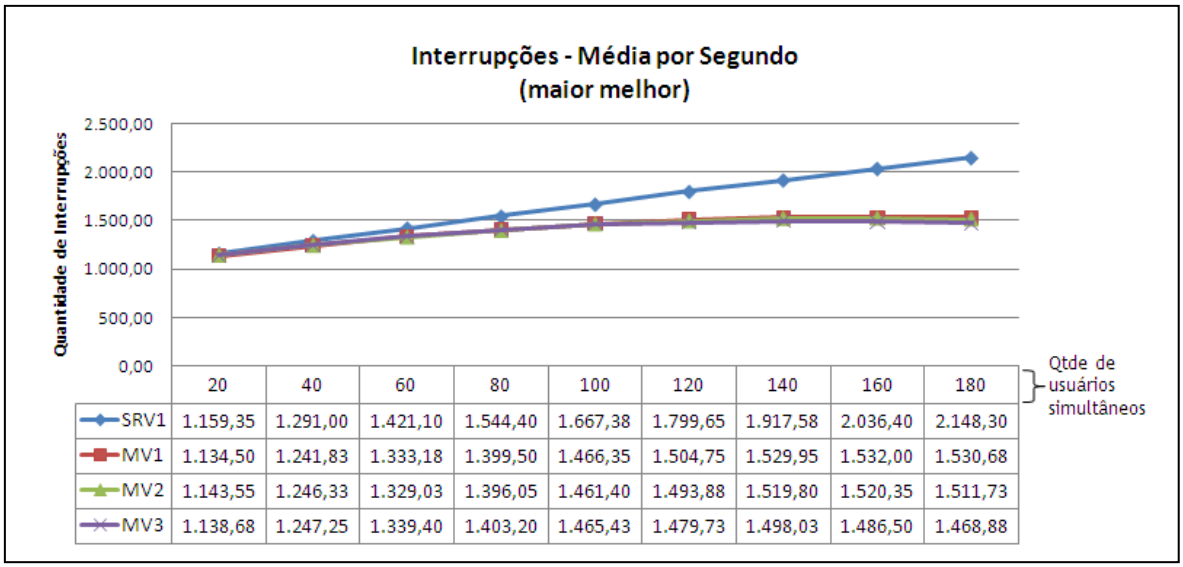
Tabela 4.12 – Estatísticas de processos e segmentos

Qtde usuários simultâneos	Servidor	Número de processos aguardando por UCP (média por segundo)	Número de processos aguardando por UCP - Normalizado em SRV1	Total processos + segmentos na lista de processos (média por segundo)	Total processos + segmentos na lista de processos - Normalizado em SRV1
20	**SRV1**	**0,18**		**236**	
	MV1	0,28	57%	213	-9,55%
	MV2	0,33	86%	215	-8,94%
	MV3	0,13	-29%	213	-9,71%
40	**SRV1**	**0,25**		**257**	
	MV1	0,60	140%	235	-8,77%
	MV2	0,38	50%	235	-8,46%
	MV3	0,58	130%	244	-5,03%
60	**SRV1**	**0,60**		**278**	
	MV1	0,83	38%	269	-3,42%
	MV2	0,68	13%	257	-7,69%
	MV3	0,78	29%	265	-4,74%
80	**SRV1**	**0,75**		**300**	
	MV1	0,98	30%	275	-8,21%
	MV2	1,20	60%	277	-7,56%
	MV3	1,15	53%	285	-4,91%
100	**SRV1**	**0,58**		**321**	
	MV1	1,95	239%	296	-7,70%
	MV2	1,65	187%	299	-6,91%
	MV3	1,65	187%	307	-4,52%
120	**SRV1**	**0,90**		**350**	
	MV1	1,35	50%	316	-9,60%
	MV2	1,58	75%	319	-8,77%
	MV3	4,20	367%	328	-6,41%
140	**SRV1**	**1,15**		**371**	
	MV1	1,90	65%	345	-6,97%
	MV2	2,98	159%	340	-8,32%
	MV3	4,80	317%	353	-4,94%
160	**SRV1**	**1,35**		**392**	
	MV1	2,60	93%	366	-6,54%
	MV2	3,70	174%	361	-7,81%
	MV3	7,35	444%	373	-4,74%
180	**SRV1**	**1,63**		**413**	
	MV1	3,63	123%	387	-6,21%
	MV2	6,50	300%	382	-7,41%
	MV3	13,30	718%	395	-4,32%

4.7 - COMPARATIVOS DO NÚMERO DE INTERRUPÇÕES

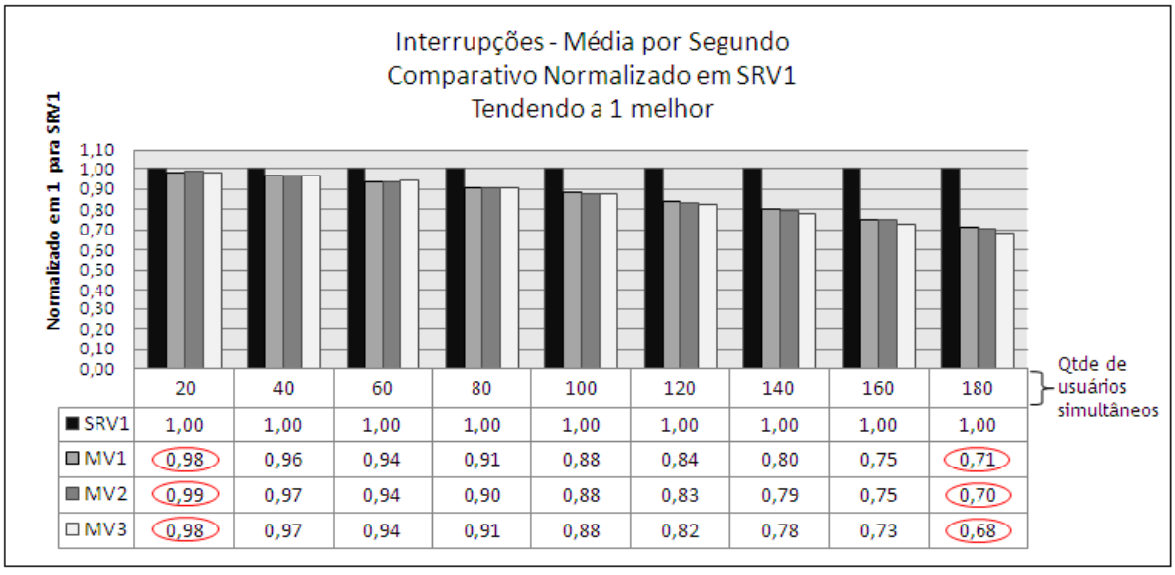
Durante a execução de um programa, alguns eventos inesperados podem ocorrer, ocasionando um desvio forçado no seu fluxo normal de execução. Esses eventos são conhecidos como interrupção, e podem ser resultado de sinalizações de algum dispositivo de *hardware* externo ao ambiente memória/processador. A interrupção é o mecanismo que permitiu a implementação da concorrência nos computadores, sendo o fundamento básico dos sistemas multiprogramáveis/multitarefa. Um exemplo de interrupção é quando um dispositivo avisa a UCP que alguma operação de E/S está completa. Neste caso, o processador deve interromper o programa em execução para tratar o término da operação sinalizada pela interrupção. (SILVA, 2010).

Através da Figura 4.32 pode-se observar que a quantidade de interrupções tratadas pelo SRV1, em todos os testes, foi superior à MV1, MV2 e MV3. Normalizando esta quantidade em SRV1, Figura 4.33, é possível apurar que a queda na quantidade de interrupções tratada em cada teste praticamente tem o mesmo valor percentual para as soluções virtualizadas. Entretanto, a eficiência de tratamento de interrupções pelo ambiente virtualizado que no início dos testes (20 usuários simultâneos) era 98% para a MV1, 99% para a MV2 e 98% para a MV3, no teste final (180 usuários) diminuiu para 71%, 70% e 68% para a MV1, MV2 e MV3, respectivamente.



	20	40	60	80	100	120	140	160	180
SRV1	1.159,35	1.291,00	1.421,10	1.544,40	1.667,38	1.799,65	1.917,58	2.036,40	2.148,30
MV1	1.134,50	1.241,83	1.333,18	1.399,50	1.466,35	1.504,75	1.529,95	1.532,00	1.530,68
MV2	1.143,55	1.246,33	1.329,03	1.396,05	1.461,40	1.493,88	1.519,80	1.520,35	1.511,73
MV3	1.138,68	1.247,25	1.339,40	1.403,20	1.465,43	1.479,73	1.498,03	1.486,50	1.468,88

Figura 4.32 – Interrupções – Média por segundo



	20	40	60	80	100	120	140	160	180
SRV1	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
MV1	0,98	0,96	0,94	0,91	0,88	0,84	0,80	0,75	0,71
MV2	0,99	0,97	0,94	0,90	0,88	0,83	0,79	0,75	0,70
MV3	0,98	0,97	0,94	0,91	0,88	0,82	0,78	0,73	0,68

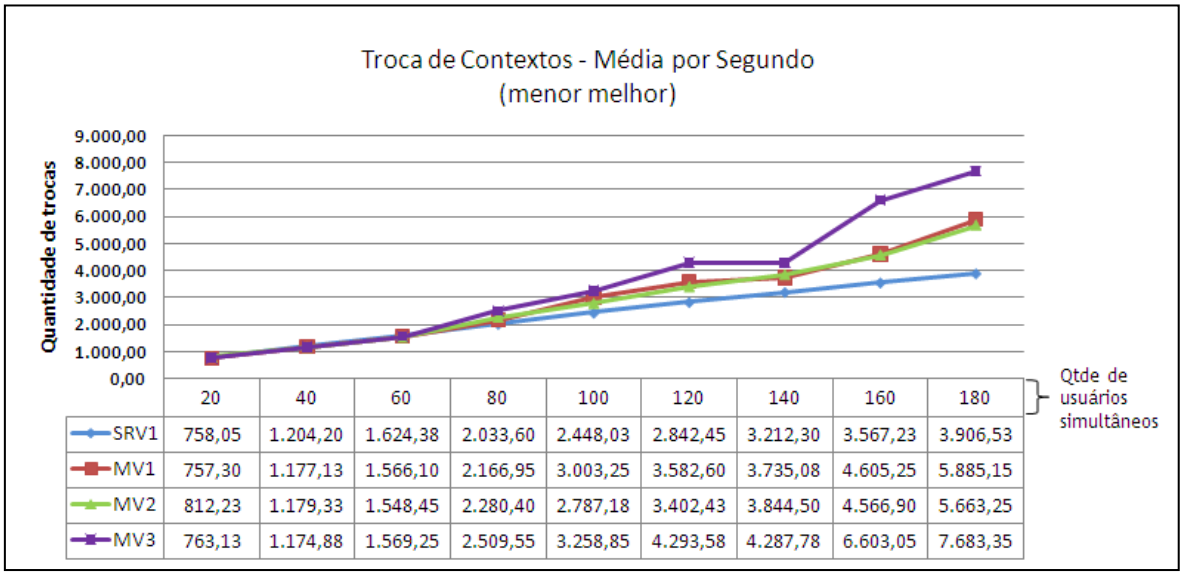
Figura 4.33 – Interrupções – Média por segundo – Normalizado em SRV1

Uma possível explicação para essa queda pode ser em decorrência do maior enfileiramento de processos nos ambientes virtualizados e à maior latência nas operações de E/S em disco. Em ambos os casos há uma janela maior para atender as necessidades do aplicativo, portanto, um maior espaçamento de tempo para geração ou sinalização de eventos acionadores de uma interrupção.

Esse resultado indica que os ambientes virtualizados apresentam um menor número de tratamento de interrupções.

4.8 - COMPARATIVOS DA QUANTIDADE DE TROCA DE CONTEXTOS

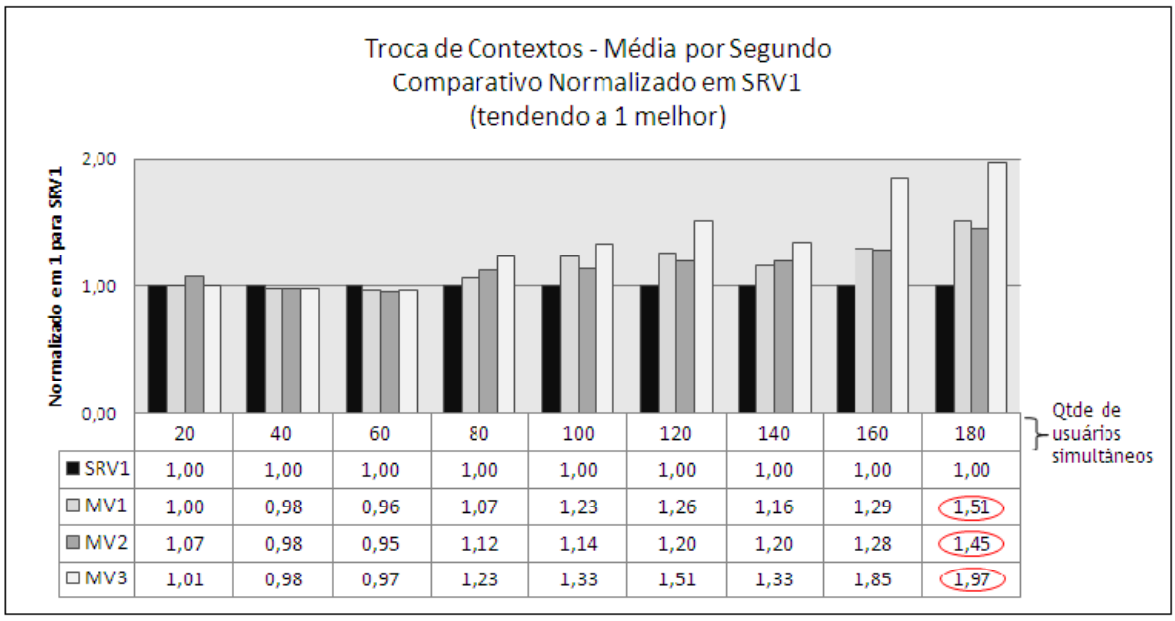
Conforme exposto anteriormente, a troca de contexto tem um alto custo porque muitas vezes a quantidade de recursos computacionais manipulados para viabilizar uma troca de contextos é maior que a quantidade de recursos desalocados e alocados para os processos que estão deixando e passando a usar o modo *kernel*, respectivamente (ATWOOD, 2008). Os ambientes virtualizados apresentaram quantidades de trocas de contextos similares até o teste de 60 usuários simultâneos conforme pode ser visto na Figura 4.34. No entanto, com 80 ou mais usuários simultâneos, os ambientes virtualizados passam a superar a quantidade do SRV1, apresentando a MV3 a maior quantidade, seguida pela MV1, MV2 e SRV1.



	20	40	60	80	100	120	140	160	180
SRV1	758,05	1.204,20	1.624,38	2.033,60	2.448,03	2.842,45	3.212,30	3.567,23	3.906,53
MV1	757,30	1.177,13	1.566,10	2.166,95	3.003,25	3.582,60	3.735,08	4.605,25	5.885,15
MV2	812,23	1.179,33	1.548,45	2.280,40	2.787,18	3.402,43	3.844,50	4.566,90	5.663,25
MV3	763,13	1.174,88	1.569,25	2.509,55	3.258,85	4.293,58	4.287,78	6.603,05	7.683,35

Figura 4.34 – Troca de contexto – Média por segundo

Normalizando a quantidade de troca de contextos em SRV1 (Figura 4.35), observa-se que quanto maior a quantidade de usuários simultâneos, maior a quantidade de trocas de contextos processadas pelos ambientes virtualizados. No teste de maior volumetria, 180 usuários simultâneos, a MV3 chega a processar 1,97 vezes mais trocas de contexto que o SRV1, a MV1 1,51 e a MV2 1,45.



	20	40	60	80	100	120	140	160	180
SRV1	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
MV1	1,00	0,98	0,96	1,07	1,23	1,26	1,16	1,29	1,51
MV2	1,07	0,98	0,95	1,12	1,14	1,20	1,20	1,28	1,45
MV3	1,01	0,98	0,97	1,23	1,33	1,51	1,33	1,85	1,97

Figura 4.35 – Troca de contexto – Média por segundo – Normalizado em SRV1

Resumindo, a quantidade média de troca de contextos dos ambientes virtualizados superou o tradicional. Com 180 usuários simultâneos, a MV1 precisou tratar 1,5 vezes mais trocas que o SRV1, a MV2 1,45 e a MV3 1,97. Esse fato indica que nos ambientes virtualizados há uma maior incidência de trocas de contexto (mudanças para o modo *kernel*).

5 - CONCLUSÕES

Este trabalho discutiu a virtualização em plataforma *x86* e abordou as 2 arquiteturas de virtualização atualmente disponíveis: via *software* e via *hardware*. Foram evidenciados para cada arquitetura os impactos da inserção da camada de virtualização para os sistemas operacionais. Como os sistemas operacionais originalmente não foram projetados para suportar a partilha dos recursos físicos de um servidor, foi preciso criar condições para que eles continuassem a acreditar que ainda possuíam o controle sobre os dispositivos físicos do *hardware*. Essas condições implicaram em passos adicionais de interação entre o sistema operacional e a solução de virtualização. A evolução das soluções de virtualização, baseadas em *software* e assistidas por *hardware*, vem provendo constantes melhorias com o intuito de reduzir, ou eliminar, a sobrecarga de processamento decorrente desses passos adicionais.

Através de um estudo comparativo de desempenho entre o ambiente tradicional e o virtualizado, aplicado a banco de dados, foram identificadas e analisadas as diferenças nos quesitos tempo de resposta e consumo de recursos computacionais do servidor de banco de dados. Esse estudo cobriu as 3 técnicas de virtualização disponíveis para a plataforma *x86*: virtualização completa sem assistência por *hardware*, paravirtualização e virtualização completa assistida por *hardware*. As métricas utilizadas para aferir a sobrecarga no ambiente virtualizado foram: (i) capacidade de tratar transações de bancos de dados; (ii) tempo de resposta da transação; (iii) consumo de UCP; (iv) uso de memória; (v) taxas de leitura e gravação em disco; (vi) capacidade de atendimento a processos e segmentos; (vii) tratamento de interrupções; e (viii) tratamento de trocas de contextos.

Os resultados obtidos permitiram identificar os seguintes aspectos inerentes ao ambiente virtualizado no que diz respeito ao desempenho da aplicação e ao consumo de recursos do servidor: (i) a eficiência de transações foi compatível com resultados obtidos por testes de laboratórios feitos por fornecedores; (ii) houve degradação do tempo de resposta das transações de banco de dados cuja intensidade aumentava à medida que se aumentava a carga de trabalho no banco de dados; (iii) o consumo de UCP em modo *kernel* foi superior; (iv) o uso de memória manteve-se nos mesmos patamares do ambiente tradicional; (v) a latência nas operações de E/S foi maior; (vi) o enfileiramento de processos aguardando por UCP foi maior; (vii) houve uma menor capacidade de tratamento de interrupções; (viii) houve maior incidência de trocas de contexto.

Diante dos resultados apurados, pode-se concluir que a adoção da virtualização para suportar um Sistema de Gestão de Base de Dados (SGBD), em plataforma *x86*, pode comprometer o objetivo básico e mandatório de desempenho. Diante dessa constatação recomenda-se cautela na hora de decidir por instalar um SGBD em ambiente virtualizado.

Entre as três técnicas testadas, não se observou uma modalidade que se destacasse significativamente em relação às demais. Pelo menos para o perfil de carga de trabalho definido para esse estudo, que é composto por muitas e pequenas transações e com leituras e escritas randômicas. Contudo, o mundo real muitas vezes permite certa tolerância para o tempo de resposta e consumo de UCP. Nos testes realizados, observou-se que cargas de trabalho menores têm menor impacto no tempo de resposta, Tabela 5.13, e no consumo de UCP, que foi o recurso físico que sofreu maior impacto em decorrência da virtualização, Tabela 5.14.

Tabela 5.13 – Resumo do tempo de resposta – Normalizado em SRV1

Qtde Usuários Simultâneos	*Customer Registration*				*Browser Product*				*Order Product*				*Process Order*				*Browse Order*			
	SRV1	MV1	MV2	MV3	SRV1	MV1	MV2	MV3	SRV1	MV1	MV2	MV3	SRV1	MV1	MV2	MV3	SRV1	MV1	MV2	MV3
20	1,00	3,00	3,00	3,00	1,00	2,63	2,63	2,25	1,00	2,36	2,42	2,21	1,00	1,67	2,00	1,67	1,00	2,50	2,50	2,00
40	1,00	3,00	3,00	3,00	1,00	2,56	2,78	2,44	1,00	2,53	2,62	2,44	1,00	2,00	2,33	2,00	1,00	2,75	2,75	2,50
60	1,00	4,00	3,00	4,00	1,00	3,11	3,11	2,56	1,00	2,77	2,86	2,51	1,00	2,67	2,67	2,33	1,00	3,00	3,25	2,75
80	1,00	4,00	6,00	6,00	1,00	3,44	4,00	3,33	1,00	3,17	3,69	3,31	1,00	3,33	4,00	4,00	1,00	3,75	4,00	3,75
100	1,00	4,00	3,00	4,50	1,00	4,00	4,33	3,78	1,00	3,92	3,89	3,58	1,00	5,00	4,67	4,33	1,00	4,75	4,75	4,75
120	1,00	4,00	5,00	4,50	1,00	4,67	5,00	4,11	1,00	4,22	4,36	3,69	1,00	4,00	4,25	3,75	1,00	4,60	4,80	4,00
140	1,00	5,00	5,00	5,00	1,00	4,78	5,00	4,78	1,00	4,16	4,32	4,22	1,00	4,50	4,50	4,50	1,00	4,80	4,80	4,60
160	1,00	6,00	6,00	10,00	1,00	5,11	5,67	6,33	1,00	4,37	4,68	5,55	1,00	5,00	5,50	8,00	1,00	5,00	5,40	6,80
180	1,00	9,00	9,50	15,50	1,00	5,80	6,50	7,60	1,00	5,68	5,76	6,53	1,00	7,75	8,00	11,25	1,00	7,00	7,40	9,80

Tabela 5.14 – Resumo do consumo médio de UCP – Normalizado em SRV1

Quantidade de Usuários Simultâneos	% Consumo Total de UCP - Média				Adicional do Consumo Normalizado em SRV1		
	SRV1	MV1	MV2	MV3	MV1	MV2	MV3
20	2,05	4,27	4,32	4,30	2,08	2,11	2,10
40	4,08	8,56	8,33	9,02	2,10	2,04	2,21
60	6,80	12,83	13,03	13,57	1,89	1,92	2,00
80	8,88	17,53	17,91	19,64	1,98	2,02	2,21
100	11,03	23,91	23,78	28,21	2,17	2,16	2,56
120	13,03	29,29	31,33	39,73	2,25	2,41	3,05
140	15,74	36,99	37,95	46,18	2,35	2,41	2,93
160	18,30	45,04	45,77	55,64	2,46	2,50	3,04
180	20,60	51,74	53,56	66,87	2,51	2,60	3,25

Diante do exposto, possivelmente bancos de dados cujos acessos não são sensíveis ao tempo de resposta, ou possuem tolerância ao tempo resposta, podem ser candidatos à virtualização. No entanto, recomenda-se, em ambos os casos, a realização de testes prévios para apurar se o perfil de acesso ao banco de dados compromete (ou não) o desempenho da aplicação e do ambiente virtualizado. Resultados que apurem que a sobrecarga da virtualização não compromete o desempenho sinalizam oportunidades para a TI adotar a virtualização e com isso reduzir ocupação de espaço dos Centros de Dados, diminuir custos e dar maior agilidade no provimento de infraestrutura para atender as necessidades e negócios da organização.

5.1 - SUGESTÕES PARA TRABALHOS FUTUROS

Trabalhos futuros, que enquadrarem no contexto de avaliar a capacidade das soluções de virtualização em plataforma *x86* para suportar a camada de banco de dados, deverão ter como meta aprofundar os estudos comparativos considerando outras ferramentas que possibilitem a geração de carga de trabalho e outras soluções de virtualização. Embora a solução *VMware* domine esse mercado desde 2001 (BITTMAN, et al., 2010), aquisições e novos investimentos têm trazido grandes fornecedores de *software* como a *Microsoft,* a *Oracle* e a *Citrix*[27] para esse mercado, com promessas de soluções que atendam o âmbito técnico-funcional e que sejam capazes de atingir capacidade de processamento e desempenho próximos ao observado no ambiente tradicional.

Além disso, sugere-se explorar a forma como as soluções de virtualização tratam as instruções privilegiadas e sensíveis, com e sem assistência via *hardware,* a fim de demonstrar tecnicamente as condições pelas quais ocorre a sobrecarga dos recursos computacionais de UCP, acesso a disco, interrupções e trocas de contextos. O resultado desta investigação pode ajudar na identificação de ajustes em configurações e parametrizações dos *hardwares* e *softwares* envolvidos que favoreçam ou contribuam para a redução da sobrecarga apurada e melhoria nos tempos de respostas.

[27] A *Citrix* é uma empresa americana multinacional que oferece soluções de virtualização, de *softwares* como serviço (*SaaS – Software as Service*) e computação em nuvens.

Outra abordagem recomendada é considerar as evoluções tecnológicas de servidores *x86*. Tecnologias como a família *Nehalem*[28], recentemente lançada pela *Intel*, podem oferecer a velocidade de processamento necessária para que a sobrecarga imposta pela camada de virtualização seja minimizada ou reduzida. Segundo o fornecedor *Intel* (PROWESS, 2010), algumas melhorias feitas em seus processadores que contribuem para essa redução são: (i) capacidade de processar mais instruções por ciclo de UCP em função dos algoritmos mais eficientes e do aumento do paralelismo; (ii) maior capacidade de processamento decorrente do fato de cada processador poder executar o dobro de segmentos através da ativação do sistema de "multi-segmentação" (em inglês *multi-threading*) simultâneo; (iii) uso de maior capacidade das áreas de *cache* compartilhadas entre os processadores, o que reduz a latência de comunicação entre os processadores.

Por fim, uma abordagem desafiadora é avaliar a possibilidade de distribuir um banco de dados entre várias MV de forma a viabilizar sua utilização no ambiente virtualizado. Essa técnica, por exemplo, é a que vem sendo recomendada pelo fornecedor *VMware* de forma a sua solução ser capaz de suportar o produto de mensageria da *Microsoft*, o *Exchange Server*[29]. Essa abordagem é duplamente desafiadora, pois requer analisar e investigar a melhor distribuição da carga de processamento entre duas ou mais MV, e analisar e investigar a melhor forma de distribuir o banco de dados. A distribuição do banco de dados pode ser homogênea ou heterogênea. Na distribuição homogênea todas as MV estarão executando o mesmo SGBD. No heterogênea as MV poderão executar SGBD diferentes. Em ambos os casos estima-se haver uma sobrecarga de processamento devido à troca de mensagens e à computação adicional que é necessária para a coordenação do SGBD entre todas as MV. Adicionalmente, será necessário atentar para as questões de replicação e fragmentação de dados, inerentes à característica de um banco de dados distribuído.

[28] *Nehalem* é o codinome do novo processador da *Intel* que, segundo a própria *Intel,* está sendo ofertado com recursos de desempenho, virtualização e confiabilidade a níveis adequados para atender aplicações com uso intensivo de dados e de missão crítica, além de ser opção para substituir servidores da plataforma *RISC.*

[29] *Exchange Server* é uma aplicação servidora de e-mails de propriedade da *Microsoft* e que pode ser instalado somente em plataforma da família *Windows Server.*

REFERÊNCIAS BIBLIOGRÁFICAS

ATWOOD, Jeff. In: ATWOOD, Jeff. **Understanding User and Kernel Mode.** 2008**.** Disponível em < http://www.codinghorror.com/blog/2008/01/understanding-user-and-kernel-mode.html>. Acesso em: 3 jun. 2008, 22:15.

BARHAM, P, et al. **Xen and the art of virtualization.** Nova Iorque: ACM Press, 2003. p. 164-177.

BITTMAN, Thomas J.; DAWSON, Philip; WEISS, George. **Magic Quadrant for x86 Server Virtualization Infrastructure.** Gartner. 2010.

BITTMAN, Thomas J.; ENCK, John. **Best Practices Before You Virtualize Your Servers**. Gartner. 2008.

CAMPBELL, Sean; JERONIMO, Michael. **Applied Virtualization Technology**. 1. ed. Intel Press, 2006. ISBN 0-9764832-3-8.

CARISSIMI, Alexandre. **Virtualização: da teoria a soluções**. In: 26º SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES E SISTEMAS DISTRIBUÍDOS. 2008. p. 174-206.

CHEN, Wei et al. DBTIM: **An Advanced Hardware Assisted Full Virtualization Architecture.** ed. IEEE Computer Society. 2008.

DESAI, Anil. 2007. **O Guia Definitivo para Gerenciamento de Plataformas Virtuais**. ed. Realtime Publishers. 2007. v. 1.

EMC; CISCO. **Business Continuity with the Cisco Unified Computing System and EMC Symmetrix VMAX**. 2010. C11-599794-00.

EMULEX. **Gb/s HBA Performance Advantages for Oracle Database**. 2008. 09-489-12/08

FAWZI, Mohamed. **Virtualization and Protection Rings (Welcome to Ring -1).** 2009. Disponível em < http://fawzi.wordpress.com/2009/05/24/virtualization-and-protection-rings-welcome-to-ring-1-part-i/.>. Acesso em 19 jul. 2009.

FELBER, Edmilosn J. W. **Proposta de uma Ferramenta OLAP em um *DATA MART* Comercial: uma aplicação prática na indústria atacadista.** 2005. 15f. Trabalho de Conclusão de Curso. Centro Universitário Feevale. Instituto de Ciências Exatas e Tecnológicas. Novo Hamburgo, 2005.

FERREIRA, Manuela Klanovicz. **Estudo e Modelagem de Instruções de Virtualização Intel VT-x para Arquitetura MIPS R3000.** 2008. Trabalho de Conclusão de Curso (graduação). Instituto de Informática, Universidade Federal do Rio Grande do Sul. Porto Alegre, 2008.

GAMMAGE, Brian; DAWSON, Philip. **Server Workloads: What Not to Virtualize.** Gartner. 2008. ID: G00156214.

GAREISS, Robin. Virtualization Expanding in the Workplace—for Good Reason. **Network World.** 1 p. 1 dez. 2009.

GARTNER. **THE 12th ANNUAL FUTURE OF IT CONFERENCE: THE ECONOMICS OF IT - Exploiting Technology to Achieve Competitive**. São Paulo, São Paulo, Brasil. 2007.

HUMPHREYS, John; GRIESER, Tim. **Mainstreaming Server Virtualization: The Intel Approach.** IDC Information and Data. Framingham, MA, USA. 2006. White Paper #201922.

KOSLOVSKI, Guilherme P.; BOUFLEUR, Marcio P.; CHAÃO, Andrea. **Uso de Virtualização de Recursos Computacionais na Administração de Redes.** Laboratório de Sistemas de Computação (LSC), Universidade Federal de Santa Maria (UFSM). Santa Maria. 2006.

LIM, Beng-Hong; LE, Bich C.; BUGNION, Edouard. **Deferred shadowing of segment descriptors in a virtual machine monitor for a segmented computer architecture**. 2004. 42 p.

MAZIERO, Carlos Alberto. **Sistemas Operacionais.** Capitulos: I-Conceitos Basicos e IX-Máquinas Virtuais. 2008. p. 32.

MENASCÉ, Daniel. A. **Virtualization: Concepts, Applications, and Performance Modeling.** 2005. 7 f. Trabalho realizado em parceria com a NGA (National Intelligence Agency), contrato NMA501-03-1-2033.

OGDEN, John Fisher. **Hardware Support for Efficient Virtualization**. San Diego, USA. 2006. 12 p.

PHELPS, John R.; DAWSON, Philip. **Desmystifying Server Virtualization Taxonomy and Terminology.** Gartner. 2007. ID: G00148373.

PINTO, Welington. **Virtualize Your Enterprise and Cut Costs With Oracle VM.** 2009. 61 p.

POPEK, G.; GOLDERG, R. **Formal requirements for virtualizable 3rd generation.** 1974. pp. 412-421.

PORTAL DOMINIC GILES. Estados Unidos, atualizado por Gilles Dominic, 2005, Apresenta dados sobre a ferramenta *Swingbench* e oferece opção para seu download. Disponível em <http://dominicgiles.com/index.html>. Acesso em 1 mai. 2010.

PORTAL ORACLE. Apresenta portal para os produtos da Oracle, incluindo opção para download do Oracle 11G. Disponível em <http://www.oracle.com/technetwork/index.html>. Acesso em 1 mai. 2010.

PORTAL RED HAT. Apresenta portal para os produtos e services da *Red Hat*, incluíndo opção para download do *Red Hat* Enterprise Linux. Disponível em < <http://www.redhat.com/rhel/details/eval/.>. Acesso em 1 abr. 2010.

PORTAL VMWARE. Apresenta portal para os produtos da VMware, incluíndo opção para download do VMware ESXi. Disponível em <http://www.vmware.com/support/>. Acesso em 1 abr. 2010.

PROWESS. **How mission-critical database workloads perform when virtualized with Intel Xeon processor 7500 series-based servers and VMware vSphere**. 2010. 23 p.

ROSE, **Robert. Survey of System Virtualization Techniques.** 8 mar. 2004. 15 p.

SILVA, Allan W. **Dados de Virtualização em Plataforma *x86* [mensagem de trabalho].** Mensagem recebida por aneiva@oi.net.br em 16 jun. 2010.

SILVA, Marcelo Pereira. **Arquitetura de Computadores – Curso Técnico de Informática.** 2010. p. 47.

THE TOLLY GROUP. **Performance Evaluation of Oracle VM Server Virtualization Server. Test Summary**.2008. ID 208322-psqifpvt1-cdb-12sep08.

VIRTUOZZO SWSOFT. **Top Ten Considerations for Chosing a Server Virtualization Technology.** Julho 2006. 7 p.

VMWARE. **VMware Sphere: Manage for Performance. Student Manual – ESX 4.0, ESXi 4.0 and vCenter Server 4.0**. Palo Alto, CA. 2010. EDU-ENG-A-MP4-LAB-STU.

_____. **Virtualizing Performance-Critical Database Applications in VMware vSphere. Performance Study.** 2009. EN-000215-00.

_____. **Performance Evaluation of Intel EPT Hardware Assist**. Palo Alto, CA. 2008-2009. EN-000191-00

_____. **Virtual Machine Monitor Execution Modes in VMware vSphere 4.0**. Palo Alto, CA. 2008.

_____. **Understanding Full Virtualization, Paravirtualization, and Hardware Assist.** Palo Alto, CA. 2007. 17 p. Item: WP-028-PRD-01-01.

_____. **Virtualizationg: Architectural Considerations And Other Evaluation Criteria.** Palo Alto, CA. 2005.

ZOVI, Dino A. Dai. **Hardware Virtualization RootKits**. Projeto ChinaShop. 2006. p. 38.

ANEXOS

ANEXO A – PROPRIEDADES DAS MÁQUINAS VIRTUAIS MV1, MV2 E MV3

Para a MV1, a opção de virtualização foi habilitada como via *software*, tanto para UCP quanto memória. Para a MV2, além dessa opção, também foi habilitada a opção de paravirtualização. A MV3 teve configurada a virtualização de UCP via *hardware* e memória via *software*. As figuras Figura A.1, Figura A.2 e Figura A.3 exibem as telas do *VMware ESXi* com as opções configuradas para atender a MV1, MV2 e MV3, respectivamente.

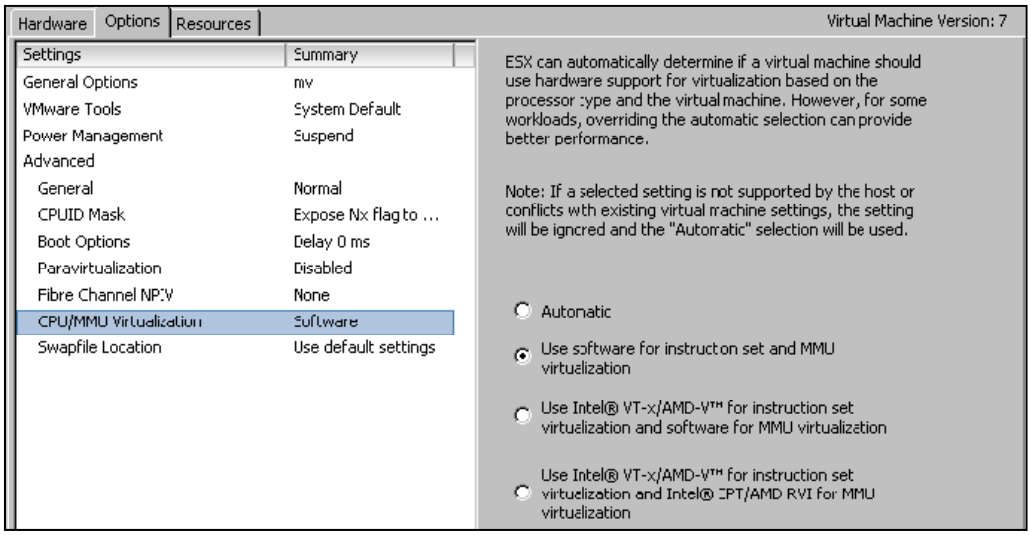


Figura A.1 – MV1 – Virtualização de UCP e memória via *software*

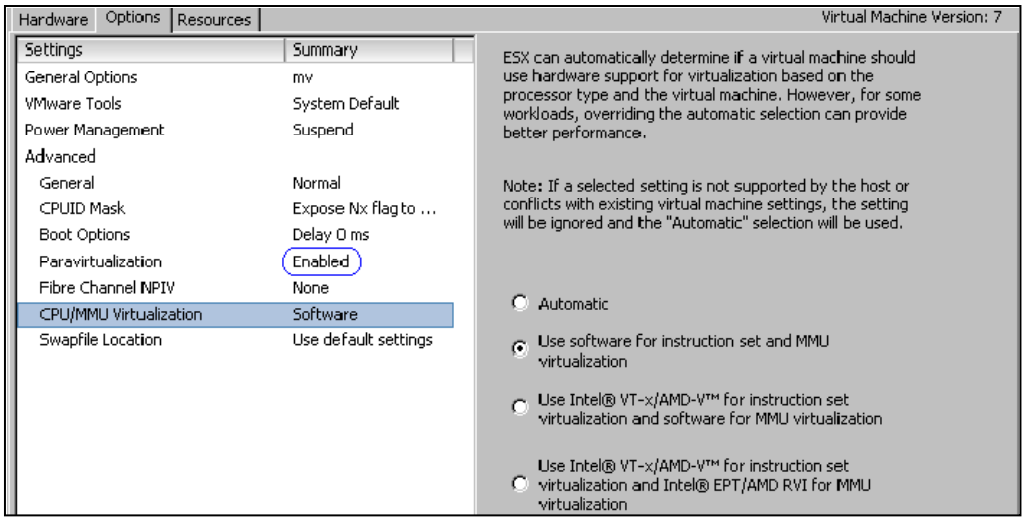


Figura A.2 – MV2 – Virtualização de UCP e memória via *software* e uso da paravirtualização

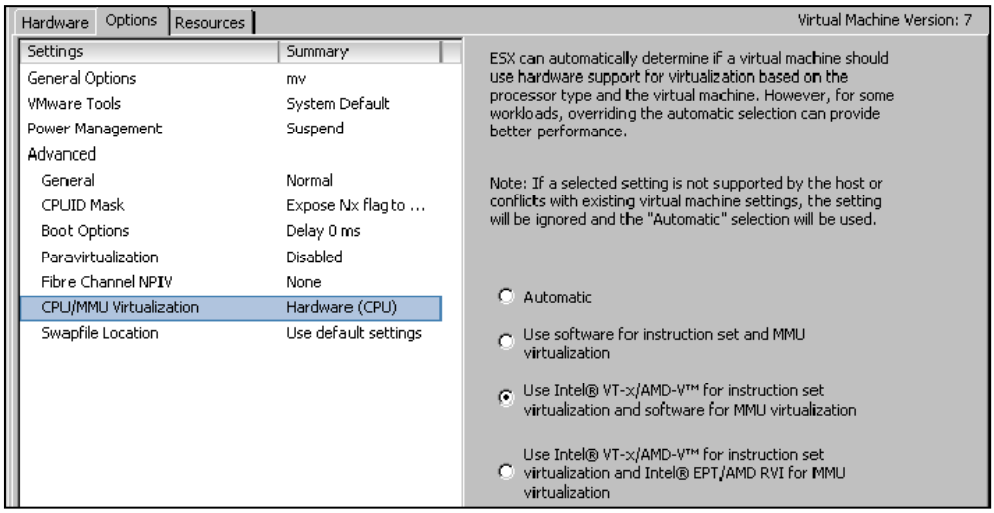


Figura A.3 – MV3 – Virtualização de UCP assistida por *hardware* e memória via *software*

As opções "*automatic*" e "*Use Intel® VT-x/AMD-V™ for instruction set virtualization and Intel® EPT/AMD RVI for MMU virtualization*", exibidas nas figuras Figura A.1, Figura A.2 e Figura A.3, não foram utilizadas por que não está homologado para o conjunto *VMware ESXi* 4.0 com processador *Intel Xeon* E5410 e LRH 5 o uso da virtualização de memória assistida por *hardware* (VMWARE, 2008).

ANEXO B – PARAMETRIZAÇÕES DO *VMWARE ESXI*

Relação dos parámetros modificados no painel *Advanced Settings* no *vSphere Client* conforme instruções passadas pela VMware e documentadas em (VMWARE, 2009)*:*

- Net.CoalesceLowRxRate: modificado o valor padrão de fábrica de 4 para 1.
- Net.CoalesceLowTxRate: modificado o valor padrão de fábrica de 4 para 1.
- Net.vmxnetThroughputWeight: modificado o valor padrão de fábrica de 0 para 128.

Os dois primeiros parâmetros estão relacionados com a taxa pela qual os pacotes de rede são agrupados para a transmissão e recepção. As modificações feitas podem melhorar em aproximadamente 1% a vazão de transmissão dos pacotes (*throughput*).

O último parâmetro favorece a transmissão sobre o tempo de resposta. Com essa mudança a vazão da transmissão pode aumentar em até aproximadamente 3%.

ANEXO C – PARAMETRIZAÇÕES DO *LINUX RED HAT*

Conteúdo do arquivo de configuração de parâmetros do *kernel* em tempo de execução, utilizado igualmente para o SRV1, MV1, MV2 e MV3:

```
# Kernel sysctl configuration file for Red Hat Linux
#
# For binary values, 0 is disabled, 1 is enabled.  See sysctl(8) and
# sysctl.conf(5) for more details.

# Controls IP packet forwarding
net.ipv4.ip_forward = 0

# Controls source route verification
net.ipv4.conf.default.rp_filter = 1

# Do not accept source routing
net.ipv4.conf.default.accept_source_route = 0

# Controls the System Request debugging functionality of the kernel
kernel.sysrq = 0

# Controls whether core dumps will append the PID to the core filename
# Useful for debugging multi-threaded applications
kernel.core_uses_pid = 1

# Controls the use of TCP syncookies
net.ipv4.tcp_syncookies = 1

# Controls the maximum size of a message, in bytes
kernel.msgmnb = 65536

# Controls the default maxmimum size of a mesage queue
kernel.msgmax = 65536

# Controls the maximum shared segment size, in bytes
kernel.shmmax = 4294967295

# Controls the maximum number of shared memory segments, in pages
kernel.shmall = 268435456

###############################################
# Valores dos parametros de kernel para Oracle
kernel.shmall = 2097152
```

```
kernel.shmmax = 17034528768
kernel.shmmni = 4096
kernel.sem = 250 32000 100 128
fs.file-max = 6553600
net.ipv4.ip_local_port_range = 1024 65000
#net.core.rmem_default = 262144
net.core.rmem_default = 4194304
#net.core.rmem_max = 262144
net.core.rmem_max = 4194304
net.core.wmem_default = 262144
net.core.wmem_max = 262144
```

ANEXO D – PARAMETRIZAÇÕES DO *ORACLE 11G R2*

Conteúdo do arquivo de configuração de parâmetros do banco de dados Oracle que seão lidos quando o Oracle é iniciado. Foi utilizado igualmente para o SRV1, MV1, MV2 e MV3:

```
#
# $Header: init.ora 06-aug-98.10:24:40 atsukerm Exp $
#
# Copyright (c) 1991, 1997, 1998 by Oracle Corporation
# NAME
#   init.ora
# FUNCTION
# NOTES
# MODIFIED
#     atsukerm   08/06/98 -  fix for 8.1.
#     hpiao      06/05/97 -  fix for 803
#     glavash    05/12/97 -  add oracle_trace_enable comment
#     hpiao      04/22/97 -  remove ifile=, events=, etc.
#     alingelb   09/19/94 -  remove vms-specific stuff
#     dpawson    07/07/93 -  add more comments regarded archive start
#     maporter   10/29/92 -  Add vms_sga_use_gblpagfile=TRUE
#     jloaiza    03/07/92 -  change ALPHA to BETA
#     danderso   02/26/92 -  change db_block_cache_protect to _db_block_cache_p
#     ghallmar   02/03/92 -  db_directory -> db_domain
#     maporter   01/12/92 -  merge changes from branch 1.8.308.1
#     maporter   12/21/91 -  bug 76493: Add control_files parameter
#     wbridge    12/03/91 -  use of %c in archive format is discouraged
#     ghallmar   12/02/91 -  add global_names=true, db_directory=us.acme.com
#     thayes     11/27/91 -  Change default for cache_clone
#     jloaiza    08/13/91 -         merge changes from branch 1.7.100.1
#     jloaiza    07/31/91 -         add debug stuff
#     rlim       04/29/91 -         removal of char_is_varchar2
#   Bridge     03/12/91 - log_allocation no longer exists
#   Wijaya     02/05/91 - remove obsolete parameters
#
###########################################################################
#####
# Example INIT.ORA file
#
# This file is provided by Oracle Corporation to help you customize
# your RDBMS installation for your site.  Important system parameters
# are discussed, and example settings given.
#
```

```
# Some parameter settings are generic to any size installation.
# For parameters that require different values in different size
# installations, three scenarios have been provided: SMALL, MEDIUM
# and LARGE.  Any parameter that needs to be tuned according to
# installation size will have three settings, each one commented
# according to installation size.
#
# Use the following table to approximate the SGA size needed for the
# three scenarious provided in this file:
#
#                -------Installation/Database Size------
#                 SMALL          MEDIUM          LARGE
#  Block        2K    4500K          6800K          17000K
#  Size         4K    5500K          8800K          21000K
#
# To set up a database that multiple instances will be using, place
# all instance-specific parameters in one file, and then have all
# of these files point to a master file using the IFILE command.
# This way, when you change a public
# parameter, it will automatically change on all instances.  This is
# necessary, since all instances must run with the same value for many
# parameters. For example, if you choose to use private rollback segments,
# these must be specified in different files, but since all gc_*
# parameters must be the same on all instances, they should be in one file.
#
# INSTRUCTIONS: Edit this file and the other INIT files it calls for
# your site, either by using the values provided here or by providing
# your own.  Then place an IFILE= line into each instance-specific
# INIT file that points at this file.
#
# NOTE: Parameter values suggested in this file are based on conservative
# estimates for computer memory availability. You should adjust values upward
# for modern machines.
#
# You may also consider using Database Configuration Assistant tool (DBCA)
# to create INIT file and to size your initial set of tablespaces based
# on the user input.
###############################################################################
######

# replace DEFAULT with your database name
db_name=DEFAULT

db_files = 80                           # SMALL
```

```
# db_files = 400                                # MEDIUM
# db_files = 1500                               # LARGE

db_file_multiblock_read_count = 8                   # SMALL
# db_file_multiblock_read_count = 16                # MEDIUM
# db_file_multiblock_read_count = 32                # LARGE

db_block_buffers = 100                              # SMALL
# db_block_buffers = 550                            # MEDIUM
# db_block_buffers = 3200                           # LARGE

shared_pool_size = 3500000                          # SMALL
# shared_pool_size = 5000000                        # MEDIUM
# shared_pool_size = 9000000                        # LARGE

log_checkpoint_interval = 10000

processes = 50                                      # SMALL
# processes = 100                                   # MEDIUM
# processes = 200                                   # LARGE

parallel_max_servers = 5                            # SMALL
# parallel_max_servers = 4 x (number of CPUs)       # MEDIUM
# parallel_max_servers = 4 x (number of CPUs)       # LARGE

log_buffer = 32768                                  # SMALL
# log_buffer = 32768                                # MEDIUM
# log_buffer = 163840                               # LARGE

# audit_trail = true            # if you want auditing
# timed_statistics = true       # if you want timed statistics
max_dump_file_size = 10240      # limit trace file size to 5 Meg each

# Uncommenting the line below will cause automatic archiving if archiving has
# been enabled using ALTER DATABASE ARCHIVELOG.
# log_archive_start = true
# log_archive_dest = disk$rdbms:[oracle.archive]
# log_archive_format = "T%TS%S.ARC"

# If using private rollback segments, place lines of the following
# form in each of your instance-specific init.ora files:
# rollback_segments = (name1, name2)

# If using public rollback segments, define how many
```

```
# rollback segments each instance will pick up, using the formula
#   # of rollback segments = transactions / transactions_per_rollback_segment
# In this example each instance will grab 40/5 = 8:
# transactions = 40
# transactions_per_rollback_segment = 5

# Global Naming -- enforce that a dblink has same name as the db it connects to
global_names = TRUE

# Edit and uncomment the following line to provide the suffix that will be
# appended to the db_name parameter (separated with a dot) and stored as the
# global database name when a database is created.  If your site uses
# Internet Domain names for e-mail, then the part of your e-mail address after
# the '@' is a good candidate for this parameter value.

# db_domain = us.acme.com # global database name is db_name.db_domain

# FOR DEVELOPMENT ONLY, ALWAYS TRY TO USE SYSTEM BACKING STORE
# vms_sga_use_gblpagfil = TRUE

# FOR BETA RELEASE ONLY.  Enable debugging modes.  Note that these can
# adversely affect performance.  On some non-VMS ports the db_block_cache_*
# debugging modes have a severe effect on performance.

#_db_block_cache_protect = true                    # memory protect buffers
#event = "10210 trace name context forever, level 2" # data block checking
#event = "10211 trace name context forever, level 2" # index block checking
#event = "10235 trace name context forever, level 1" # memory heap checking
#event = "10049 trace name context forever, level 2" # memory protect cursors

# define parallel server (multi-instance) parameters
#ifile = ora_system:initps.ora

# define two control files by default
control_files = (ora_control1, ora_control2)

# Uncomment the following line if you wish to enable the Oracle Trace product
# to trace server activity.  This enables scheduling of server collections
# from the Oracle Enterprise Manager Console.
# Also, if the oracle_trace_collection_name parameter is non-null,
# every session will write to the named collection, as well as enabling you
# to schedule future collections from the console.

# oracle_trace_enable = TRUE
```

```
# Uncomment the following line, if you want to use some of the new 8.1
# features. Please remember that using them may require some downgrade
# actions if you later decide to move back to 8.0.

#compatible = 8.1.0
```

ANEXO E – SISTEMAS DE ARQUIVOS UTILIZADOS PELOS *SOFTWARES*

Os parágrafos abaixo relacionam as partições e sistemas de arquivos utilizados nos dois servidores físicos que atenderam o SRV1, MV1, MV2 e MV3. No SRV1 tem a partição *dm-6*, sistema de arquivo */orabininst* que não existe na MV1, MV2 e MV3. Esse sistema de arquivo foi utilizado somente para baixar o binário do Oracle 11G R2. Após finalizada a baixa, o binário foi instalação no SRV1 e */orabininst* não foi mais utilizado.

Partições (*partitions*) e sistemas de arquivos (*file systems*) do SRV1:

major minor #blocks name

8	0	142737408	sda
8	1	200781	sda1
8	2	25165822	sda2
8	3	8385930	sda3
8	4	1	sda4
8	5	108976896	sda5
253	0	10485760	dm-0
253	1	6291456	dm-1
253	2	3145728	dm-2
253	3	2097152	dm-3
253	4	10485760	dm-4
253	5	6291456	dm-5
253	6	10485760	dm-6

Filesystem	**Size**	**Used**	**Avail**	**Use%**	**Mounted on**
/dev/sda3	7.8G	576M	6.8G	8%	/
/dev/mapper/vglocal-lvoradata	9.7G	4.0G	5.3G	44%	/oradata
/dev/mapper/vglocal-lvusr	5.9G	2.6G	3.0G	47%	/usr
/dev/mapper/vglocal-lvopt	3.0G	70M	2.7G	3%	/opt
/dev/mapper/vglocal-lvtmp	2.0G	72M	1.8G	4%	/tmp
/dev/mapper/vglocal-lvoracle	9.7G	3.7G	5.6G	40%	/oracle
/dev/mapper/vglocal-lvvar	5.9G	296M	5.3G	6%	/var
/dev/sda1	190M	12M	169M	7%	/boot
tmpfs	16G	2.0G	14G	13%	/dev/shm
/dev/mapper/vglocal-lvorabininst	9.9G	3.7G	5.7G	40%	/orabininst

Partições (*partitions*) e sistemas de arquivos (*file systems*) do MV1, MV2 e MV3:

major minor #blocks name

8	0	72351744	sda

8	1	200781	sda1
8	2	25165822	sda2
8	3	8385930	sda3
8	4	1	sda4
8	5	38596131	sda5
253	0	10158080	dm-0
253	1	2097152	dm-1
253	2	6291456	dm-2
253	3	10485760	dm-3
253	4	6291456	dm-4
253	5	3244032	dm-5

Filesystem	**Size**	**Used**	**Avail**	**Use%**	**Mounted on**
/dev/sda3	7.8G	4.1G	3.3G	56%	/
/dev/mapper/VolGroup00-lvoradata	9.4G	2.6G	6.4G	29%	/oradata
/dev/mapper/VolGroup00-lvtmp	2.0G	116M	1.8G	7%	/tmp
/dev/mapper/VolGroup00-lvusr	5.9G	1.9G	3.7G	35%	/usr
/dev/mapper/VolGroup00-lvoracle	9.7G	3.6G	5.7G	39%	/oracle
/dev/mapper/VolGroup00-lvvar	5.9G	205M	5.4G	4%	/var
/dev/mapper/VolGroup00-lvopt	3.0G	70M	2.8G	3%	/opt
/dev/sda1	190M	12M	169M	7%	/boot
tmpfs	16G	0	16G	0%	/dev/shm

ANEXO F – PROGRAMAS FONTES

Os parágrafos abaixo reproduzem o fonte dos seis programas utilizados para execução das cinco transações pré-definidas do *Order Entry (Customer Registration*, *Browse Products*, *Order Product*, *Process Order* e *Browse Order*s).

Programa BrowseAndUpdateOrders.java

```
package com.dom.benchmarking.swingbench.transactions;

import com.dom.benchmarking.swingbench.event.JdbcTaskEvent;
import com.dom.benchmarking.swingbench.kernel.SwingBenchException;
import com.dom.benchmarking.swingbench.kernel.SwingBenchTask;
import com.dom.benchmarking.swingbench.utilities.RandomGenerator;


import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;


import java.util.Map;


public class BrowseAndUpdateOrders extends OrderEntryProcess {
  private PreparedStatement ordPs = null;

  public BrowseAndUpdateOrders() {
  }

  public void init(Map params) {
  }

  public void execute(Map params) throws SwingBenchException {
    Connection connection   = (Connection)
params.get(SwingBenchTask.JDBC_CONNECTION);
    int        custID       =
RandomGenerator.randomInteger(MIN_CUSTID, MAX_CUSTID);
    ResultSet  rs           = null;
    initJdbcTask();

    long executeStart = System.nanoTime();

    try {
      try {
        logon(connection, custID);
        addInsertStatements(1);
        addCommitStatements(1);
        getCustomerDetails(connection, custID);
        addSelectStatements(1);
        thinkSleep();
        ordPs = connection.prepareStatement(
            " SELECT /*+ use_nl */ o.order_id, line_item_id,
product_id, unit_price, quantity, order_mode, order_status,
```

```
order_total, sales_rep_id, promotion_id, c.customer_id,
cust_first_name, cust_last_name, credit_limit, cust_email   FROM
orders o , order_items oi, customers c  WHERE o.order_id =
oi.order_id  and o.customer_id = c.customer_id  and c.customer_id =
?");
        ordPs.setInt(1, custID);
        rs = ordPs.executeQuery();
        rs.next();
        addSelectStatements(1);
      } catch (SQLException se) {
        throw new SwingBenchException(se.getMessage());
      } finally {
        try  {
          rs.close();
          ordPs.close();
        } catch (Exception ex)  {
          ex.printStackTrace();
        } finally  {
        }

      }

      processTransactionEvent(new JdbcTaskEvent(this, getId(),
(System.nanoTime() - executeStart), true, getInfoArray()));
    } catch (SwingBenchException sbe) {
      processTransactionEvent(new JdbcTaskEvent(this, getId(),
(System.nanoTime() - executeStart), false, getInfoArray()));
      throw new SwingBenchException(sbe.getMessage());
    }
  }

  public void close() {
  }
}
```

Programa BrowseProducts.java

```
package com.dom.benchmarking.swingbench.transactions;

import com.dom.benchmarking.swingbench.event.JdbcTaskEvent;
import com.dom.benchmarking.swingbench.kernel.SwingBenchException;
import com.dom.benchmarking.swingbench.kernel.SwingBenchTask;
import com.dom.benchmarking.swingbench.utilities.RandomGenerator;


import java.sql.Connection;
import java.sql.SQLException;


import java.util.Map;
import com.protomatter.syslog.Syslog;;


public class BrowseProducts extends OrderEntryProcess {
  private int custID = 0;
```

```
  public BrowseProducts() {
  }

  public void init(Map params) {
  }

  public void execute(Map params) throws SwingBenchException {

    Connection connection   = (Connection)
params.get(SwingBenchTask.JDBC_CONNECTION);
    initJdbcTask();

    long executeStart = System.nanoTime();

    try {
      try {
        custID = RandomGenerator.randomInteger(MIN_CUSTID,
MAX_CUSTID);
        logon(connection, custID);
        addInsertStatements(1);
        addCommitStatements(1);
        getCustomerDetails(connection, custID);
        addSelectStatements(1);
        thinkSleep();

        int numOfBrowseCategorys = RandomGenerator.randomInteger(1,
MAX_BROWSE_CATEGORY);

        for (int i = 0; i < numOfBrowseCategorys; i++) {
          getProductDetailsByCategory(connection,
RandomGenerator.randomInteger(MIN_CATEGORY, MAX_CATEGORY));
          addSelectStatements(1);
          thinkSleep();
        }

        numOfBrowseCategorys = RandomGenerator.randomInteger(1,
MAX_BROWSE_CATEGORY);

        for (int i = 0; i < numOfBrowseCategorys; i++) {
          getProductQuantityByCategory(connection,
RandomGenerator.randomInteger(MIN_CATEGORY, MAX_CATEGORY));
          addSelectStatements(1);
          thinkSleep();
        }
      } catch (SQLException se) {
        se.printStackTrace();
        throw new SwingBenchException(se.getMessage());
      }

      processTransactionEvent(new JdbcTaskEvent(this, getId(),
(System.nanoTime() - executeStart), true, getInfoArray()));
    } catch (SwingBenchException sbe) {
      processTransactionEvent(new JdbcTaskEvent(this, getId(),
(System.nanoTime() - executeStart), false, getInfoArray()));
      throw new SwingBenchException(sbe.getMessage());
    }
  }
```

```
  public void close() {
  }
}
```

Programa NewCustomerProcess.java

```
package com.dom.benchmarking.swingbench.transactions;

import com.dom.benchmarking.swingbench.event.JdbcTaskEvent;
import com.dom.benchmarking.swingbench.kernel.SwingBenchException;
import com.dom.benchmarking.swingbench.kernel.SwingBenchTask;
import com.dom.benchmarking.swingbench.utilities.RandomGenerator;


import com.protomatter.syslog.Syslog;


import java.io.BufferedReader;
import java.io.FileReader;


import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;


import java.util.ArrayList;
import java.util.Map;
import java.util.StringTokenizer;


import oracle.jdbc.OracleConnection;


public class NewCustomerProcess extends OrderEntryProcess {

  private static final String NAMES_FILE = "data/names.txt";
  private static final String NLS_FILE = "data/nls.txt";
  private static ArrayList firstNames = null;
  private static ArrayList lastNames = null;
  private static ArrayList nlsInfo = null;
  private static Object lock = new Object();
  private PreparedStatement insPs = null;
  private PreparedStatement seqPs = null;

  public NewCustomerProcess() {
  }

  public void init(Map params) throws SwingBenchException {
    boolean initCompleted = false;

    if ((firstNames == null) || !initCompleted) { // load any data
you might need (in this case only once)
```

```
      synchronized (lock) {
        if (firstNames == null) {
          firstNames = new ArrayList();
          lastNames = new ArrayList();
          nlsInfo = new ArrayList();

          String value = (String)params.get("SOE_NAMESDATA_LOC");

          try {
            BufferedReader br = new BufferedReader(new
FileReader((value == null) ? NAMES_FILE : value));
            String data = null;
            String firstName = null;
            String lastName = null;

            while ((data = br.readLine()) != null) {
              StringTokenizer st = new StringTokenizer(data, ",");
              firstName = st.nextToken();
              lastName = st.nextToken();
              firstNames.add(firstName);
              lastNames.add(lastName);
            }

            br.close();
            value = (String)params.get("SOE_NLSDATA_LOC");
            br = new BufferedReader(new FileReader((value == null) ?
NLS_FILE : value));
            data = null;

            while ((data = br.readLine()) != null) {
              NLSSupport nls = new NLSSupport();
              StringTokenizer st = new StringTokenizer(data, ",");
              nls.language = st.nextToken();
              nls.territory = st.nextToken();
              nlsInfo.add(nls);
            }

            br.close();
          } catch (java.io.FileNotFoundException fne) {
            Syslog.error(NewCustomerProcess.class, fne);
          } catch (java.io.IOException ioe) {
            Syslog.error(NewCustomerProcess.class, ioe);
          }
        }

        initCompleted = true;
      }
    }
  }

  public void execute(Map params) throws SwingBenchException {
    Connection connection =
(Connection)params.get(SwingBenchTask.JDBC_CONNECTION);
    int custID;
    String firstName =
(String)firstNames.get(RandomGenerator.randomInteger(0,
firstNames.size()));
```

```
    String lastName =
(String)lastNames.get(RandomGenerator.randomInteger(0,
lastNames.size()));
    NLSSupport nls =
(NLSSupport)nlsInfo.get(RandomGenerator.randomInteger(0,
nlsInfo.size()));
    initJdbcTask();

    long executeStart = System.nanoTime();
    ResultSet rs = null;
    try {
      try {
        seqPs = connection.prepareStatement("select
customer_seq.nextval from dual");

        rs = seqPs.executeQuery();
        rs.next();
        custID = rs.getInt(1);

        addSelectStatements(1);
        thinkSleep();

        try {
          insPs = connection.prepareStatement("insert into customers
(customer_id ,cust_first_name ,cust_last_name ,nls_language
,nls_territory ,credit_limit ,cust_email ,account_mgr_id ) " +
"values (? , ? , ? , ? , ? , ? ,? , ?) ");
          insPs.setInt(1, custID);
          insPs.setString(2, firstName);
          insPs.setString(3, lastName);
          insPs.setString(4, nls.language);
          insPs.setString(5, nls.territory);
          insPs.setInt(6,
RandomGenerator.randomInteger(MIN_CREDITLIMIT, MAX_CREDITLIMIT));
          insPs.setString(7, firstName + "." + lastName + "@" +
"oracle.com");
          insPs.setInt(8, RandomGenerator.randomInteger(MIN_SALESID,
MAX_SALESID));
          insPs.execute();

        } catch (SQLException se) {
          throw new SwingBenchException(se);
        }
        addInsertStatements(1);
        connection.commit();
        addCommitStatements(1);
        thinkSleep();
        logon(connection, custID);
        addInsertStatements(1);
        addCommitStatements(1);
        getCustomerDetails(connection, custID);
        addSelectStatements(1);
      } catch (SQLException se) {
        throw new SwingBenchException(se.getMessage());
      } finally {
        try {
          rs.close();
          seqPs.close();
```

```
          insPs.close();
        } catch (Exception ex) {
        }

      }

      processTransactionEvent(new JdbcTaskEvent(this, getId(),
(System.nanoTime() - executeStart), true, getInfoArray()));
    } catch (SwingBenchException sbe) {
      processTransactionEvent(new JdbcTaskEvent(this, getId(),
(System.nanoTime() - executeStart), false, getInfoArray()));

      try {
        connection.rollback();
      } catch (SQLException er) {
        return;
      }

      throw new SwingBenchException(sbe);
    }
  }

  public void close() {
  }

  public void populate(int numToPopulate, Connection connection)
throws SQLException { // used for initial population
    ((OracleConnection)connection).setDefaultExecuteBatch(100);

    int custID;
    String firstName =
(String)firstNames.get(RandomGenerator.randomInteger(0,
firstNames.size()));
    String lastName =
(String)lastNames.get(RandomGenerator.randomInteger(0,
lastNames.size()));
    NLSSupport nls =
(NLSSupport)nlsInfo.get(RandomGenerator.randomInteger(0,
nlsInfo.size()));
    seqPs = connection.prepareStatement("select customer_seq.nextval
from dual");
    insPs = connection.prepareStatement("insert into customers
(customer_id ,cust_first_name ,cust_last_name ,nls_language
,nls_territory ,credit_limit ,cust_email ,account_mgr_id ) " +
"values (? , ? , ? , ? , ? , ? ,? , ?) ");

    ResultSet rs = seqPs.executeQuery();
    rs.next();
    custID = rs.getInt(1);

    for (int i = 0; i < numToPopulate; i++) {
      firstName =
(String)firstNames.get(RandomGenerator.randomInteger(0,
firstNames.size()));
      lastName =
(String)lastNames.get(RandomGenerator.randomInteger(0,
lastNames.size()));
```

```
      nls = (NLSSupport)nlsInfo.get(RandomGenerator.randomInteger(0,
nlsInfo.size()));

      rs.close();
      insPs.setInt(1, custID++);
      insPs.setString(2, firstName);
      insPs.setString(3, lastName);
      insPs.setString(4, nls.language);
      insPs.setString(5, nls.territory);
      insPs.setInt(6, RandomGenerator.randomInteger(MIN_CREDITLIMIT,
MAX_CREDITLIMIT));
      insPs.setString(7, firstName + "." + lastName + "@" +
"oracle.com");
      insPs.setInt(8, RandomGenerator.randomInteger(MIN_SALESID,
MAX_SALESID));
      insPs.execute();

      if ((i % 10000) == 0) {
        connection.commit();
      }
    }
    Statement st = connection.createStatement();
    st.execute("alter sequence customer_seq increment by " +
numToPopulate);
    rs = seqPs.executeQuery();
    st.execute("alter sequence customer_seq increment by 1");

    connection.commit();

    ((OracleConnection)connection).setDefaultExecuteBatch(1);

    seqPs.close();
    insPs.close();
  }

  private class NLSSupport {

    String language = null;
    String territory = null;

  }

}
```

Programa NewOrderProcess.java

```
package com.dom.benchmarking.swingbench.transactions;

import com.dom.benchmarking.swingbench.event.JdbcTaskEvent;
import com.dom.benchmarking.swingbench.kernel.SwingBenchException;
import com.dom.benchmarking.swingbench.kernel.SwingBenchTask;
import com.dom.benchmarking.swingbench.utilities.RandomGenerator;


import com.protomatter.syslog.Syslog;
```

```
import java.io.BufferedReader;
import java.io.FileReader;


import java.sql.Connection;
import java.sql.Date;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;


import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.StringTokenizer;


import oracle.jdbc.OracleConnection;


public class NewOrderProcess extends OrderEntryProcess {

  private static final String PRODUCTS_FILE = "data/productids.txt";
  private static final int STATICLINEITEMSIZE = 3;
  private static ArrayList products;
  private static Object lock = new Object();
  private PreparedStatement insIPs = null;
  private PreparedStatement insOPs = null;
  private PreparedStatement seqPs = null;
  private PreparedStatement updIns = null;
  private PreparedStatement updOPs = null;
  private int custID;
  private int orderID;

  public NewOrderProcess() {
  }

  public void init(Map params) throws SwingBenchException {
    Connection connection =
(Connection)params.get(SwingBenchTask.JDBC_CONNECTION);
    boolean initCompleted = false;

    if ((products == null) || !initCompleted) { // load any data you
might need (in this case only once)

      synchronized (lock) {
        if (products == null) {
          products = new ArrayList();

          String value = (String)params.get("SOE_PRODUCTSDATA_LOC");

          try {
            BufferedReader br = new BufferedReader(new
FileReader((value == null) ? PRODUCTS_FILE : value));
            String data = null;
```

```
            while ((data = br.readLine()) != null) {
              StringTokenizer st = new StringTokenizer(data);
              products.add(new Integer(st.nextToken()));
            }

            br.close();
          } catch (java.io.FileNotFoundException fne) {
            Syslog.error(NewOrderProcess.class, fne);
          } catch (java.io.IOException ioe) {
            Syslog.error(NewOrderProcess.class, ioe);
          }

          try {
            this.getMaxandMinCustID(connection);
            this.getMaxandMinWarehouseID(connection);
          } catch (SQLException se) {
            Syslog.error(NewOrderProcess.class, se);
          }
        }

        initCompleted = true;
      }
    }
  }

  public void execute(Map params) throws SwingBenchException {
    List<Integer> products = null;
    Connection connection =
(Connection)params.get(SwingBenchTask.JDBC_CONNECTION);
    initJdbcTask();

    long executeStart = System.nanoTime();

    try {
      try {
        custID = RandomGenerator.randomInteger(MIN_CUSTID,
MAX_CUSTID);
        logon(connection, custID);
        addInsertStatements(1);
        addCommitStatements(1);
        getCustomerDetails(connection, custID);
        addSelectStatements(1);
        thinkSleep();

        int numOfBrowseCategorys = RandomGenerator.randomInteger(1,
MAX_BROWSE_CATEGORY);

        for (int i = 0; i < numOfBrowseCategorys; i++) { // Look at a
randomn number of products
          products = getProductDetailsByCategory(connection,
RandomGenerator.randomInteger(MIN_CATEGORY, MAX_CATEGORY));
          addSelectStatements(1);
          thinkSleep();
        }
        if (products.size() > 0) {
          ResultSet rs = null;
          try {
```

```
            seqPs = connection.prepareStatement("select
orders_seq.nextval from dual");
            rs = seqPs.executeQuery();
            rs.next();
            orderID = rs.getInt(1);
          } catch (Exception se) {
            Syslog.debug(this, "Getting Sequence : ", se);
            throw new SwingBenchException(se);
          } finally {
            hardClose(rs);
            hardClose(seqPs);
          }
          addSelectStatements(1);
          thinkSleep();

          try { //insert a order, I'd usually use a returning clause
but generic jdbc doesn't support this
            insOPs = connection.prepareStatement("insert into
orders(ORDER_ID, ORDER_DATE, CUSTOMER_ID) " + "values (?, ?, ?)");
            insOPs.setInt(1, orderID);
            insOPs.setDate(2, new Date(System.currentTimeMillis()));
            insOPs.setInt(3, custID);
            insOPs.execute();
          } catch (Exception se) {
            Syslog.debug(this, "Inserting order : ", se);
            throw new SwingBenchException(se);
          } finally {
            hardClose(insOPs);
          }

          addInsertStatements(1);
          thinkSleep();

          int numOfProductsToBuy =
RandomGenerator.randomInteger(MIN_PRODS_TO_BUY, MAX_PRODS_TO_BUY);
          double totalOrderCost = 0;
          ArrayList inventoryUpdates = new
ArrayList(numOfProductsToBuy);

          for (int lineItemID = 0; lineItemID < numOfProductsToBuy;
lineItemID++) {
            //int prodID =
((Integer)products.get(RandomGenerator.randomInteger(0,
products.size()))).intValue();
            int prodID = products.get(lineItemID);
            int quantity;
            double price;
            price = getProductDetailsByID(connection, prodID); // get
a products details
            addSelectStatements(1);
            thinkSleep();
            quantity = getProductQuantityByID(connection, prodID); //
check to see if its in stock
            addSelectStatements(1);
            thinkSleep();

            if (quantity > 0) {
              try {
```

```
                insIPs = connection.prepareStatement("insert into
order_items(ORDER_ID, LINE_ITEM_ID, PRODUCT_ID, UNIT_PRICE, QUANTITY)
" + "values (?, ?, ?, ?, ?)");
                insIPs.setInt(1, orderID);
                insIPs.setInt(2, lineItemID);
                insIPs.setInt(3, prodID);
                insIPs.setDouble(4, price);
                insIPs.setInt(5, 1);
                insIPs.execute();
              } catch (Exception se) {
                Syslog.debug(this, "Inserting Order Item : ", se);
                Syslog.debug(this, "Exception inserting lineitem
(order_id, lineItem_id) : (" + orderID + ", " + lineItemID);
                throw new SwingBenchException(se);
              } finally {
                hardClose(insIPs);
              }

              addInsertStatements(1);
            }

            thinkSleep();

            InventoryUpdate inventoryUpdate = new
InventoryUpdate(prodID,
RandomGenerator.randomInteger(MIN_WAREHOUSE_ID, MAX_WAREHOUSE_ID),
1);
            inventoryUpdates.add(inventoryUpdate);
            totalOrderCost = totalOrderCost + price;
          }

          try { //update the order
            updOPs = connection.prepareStatement("update orders " +
"set order_mode = ?, " + "order_status = ?, " + "order_total = ? " +
"where order_id = ?");
            updOPs.setString(1, "online");
            updOPs.setInt(2, RandomGenerator.randomInteger(0,
AWAITING_PROCESSING));
            updOPs.setDouble(3, totalOrderCost);
            updOPs.setInt(4, orderID);
            updOPs.execute();
          } catch (SQLException se) {
            Syslog.debug(this, "Updating Order : ", se);
            throw new SwingBenchException(se);
          } finally {
            hardClose(updOPs);
          }

          addUpdateStatements(1);
          thinkSleep();
          getOrderDetailsByOrderID(connection, orderID);
          addSelectStatements(1);
          thinkSleep();

          for (int i = 0; i < inventoryUpdates.size(); i++) {
            InventoryUpdate inventoryUpdate =
(InventoryUpdate)inventoryUpdates.get(i);
```

```
            try { //update the stock levels
              updIns = connection.prepareStatement("update
inventories " + "set quantity_on_hand = quantity_on_hand - ? " +
"where product_id = ? " + "and warehouse_id = ?");
              updIns.setInt(1, inventoryUpdate.quantityOrdered);
              updIns.setInt(2, inventoryUpdate.productID);
              updIns.setInt(3, inventoryUpdate.warehouseID);
              updIns.execute();
              updIns.close();
              addUpdateStatements(1);
            } catch (SQLException se) {
              Syslog.debug(this, "Updating inventory : ", se);
              throw new SwingBenchException(se);
            } finally {
              hardClose(updIns);
            }
          }


          connection.commit();
          addCommitStatements(1);
        }
      } catch (SQLException e) {
        Syslog.debug(this, "Unexpected Exception in NewOrderProcess()
: ", e);
        throw new SwingBenchException(e); // shouldn't happen
      }

      processTransactionEvent(new JdbcTaskEvent(this, getId(),
(System.nanoTime() - executeStart), true, getInfoArray()));
    } catch (SwingBenchException sbe) {
      processTransactionEvent(new JdbcTaskEvent(this, getId(),
(System.nanoTime() - executeStart), false, getInfoArray()));

      try {
        addRollbackStatements(1);
        connection.rollback();

      } catch (SQLException er) {
      }

      throw new SwingBenchException(sbe);
    }
  }

  public void populate(int numToPopulate, boolean isRandomSizes,
Connection connection) throws SQLException { // used for initial
population

    ((OracleConnection)connection).setDefaultExecuteBatch(100);
    seqPs = connection.prepareStatement("select orders_seq.nextval
from dual"); //insert a order
    insOPs = connection.prepareStatement("insert into
orders(ORDER_ID, ORDER_DATE, ORDER_TOTAL, CUSTOMER_ID) " + "values
(?, SYSTIMESTAMP, ?, ?)");
    insIPs = connection.prepareStatement("insert into
order_items(ORDER_ID, LINE_ITEM_ID, PRODUCT_ID, UNIT_PRICE, QUANTITY)
" + "values (?, ?, ?, ?, ?)");
```

```
    ResultSet rs = seqPs.executeQuery();
    rs.next();
    orderID = rs.getInt(1);
    rs.close();

    Map<Integer, Integer> selectedProducts = null;

    for (int x = 0; x < numToPopulate; x++) {
      int numOfProductsToBuy = (isRandomSizes) ?
RandomGenerator.randomInteger(MIN_PRODS_TO_BUY, MAX_PRODS_TO_BUY) + 1
: STATICLINEITEMSIZE;
      double totalOrderCost = 0;
      selectedProducts = new HashMap<Integer,
Integer>(numOfProductsToBuy);
      for (int i = 1; i < numOfProductsToBuy; i++) {
        Integer randProdID = null;
        while (true) {
          randProdID = RandomGenerator.randomInteger(MIN_PROD_ID,
MAX_PROD_ID);
          if (selectedProducts.get(randProdID) == null) {
            selectedProducts.put(randProdID, randProdID);
            break;
          }
        }

        //        int prodID =
((Integer)products.get(randProdID.intValue())).intValue();
        //        int prodID =
RandomGenerator.randomInteger(MIN_PROD_ID, MAX_PROD_ID);
        int quantity = 1;

        double price = RandomGenerator.randomInteger(2, 15);
        totalOrderCost += price;
        insIPs.setInt(1, orderID);
        insIPs.setInt(2, i);
        insIPs.setInt(3, randProdID);
        insIPs.setDouble(4, price);
        insIPs.setInt(5, quantity);
        insIPs.executeUpdate();
      }

      custID = RandomGenerator.randomInteger(MIN_CUSTID, MAX_CUSTID);
      insOPs.setInt(1, orderID);
      //insOPs.setDate(2, new Date(System.nanoTime()));
      insOPs.setDouble(2, totalOrderCost);
      insOPs.setInt(3, custID);
      insOPs.execute();

      if ((orderID % 10000) == 0) {
        connection.commit();
      }

      orderID++;
    }

    Statement st = connection.createStatement();
    st.execute("alter sequence orders_seq increment by " +
numToPopulate);
```

```
    rs = seqPs.executeQuery();
    st.execute("alter sequence orders_seq increment by 1");
    st.close();
    seqPs.close();
    insOPs.close();
    insIPs.close();
    ((OracleConnection)connection).setDefaultExecuteBatch(1);
  }

  public void close() {
  }

  class InventoryUpdate {

    int productID;
    int quantityOrdered;
    int warehouseID;

    public InventoryUpdate(int pid, int wid, int qo) {
      productID = pid;
      warehouseID = wid;
      quantityOrdered = qo;
    }

  }

}
```

Programa OrderEntryProcess.java

```
package com.dom.benchmarking.swingbench.transactions;

import com.dom.RandomUtilities;
import com.dom.benchmarking.swingbench.kernel.DatabaseTransaction;


import java.sql.Connection;
import java.sql.Date;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;


import java.util.ArrayList;
import java.util.List;


public abstract class OrderEntryProcess extends DatabaseTransaction {

  static final String NEW_CUSTOMER = "New Customer";
  static final String CUTOMER_ORDER = "Customer Order";
  static final String CUSTOMER_BROWSE = "Customer Browse";
  static final String GET_CUSTOMER_SEQ_TX = "Get Customer Sequence";
  static final String GET_ORDER_SEQ_TX = "Get Order Sequence";
  static final String INSERT_CUSTOMER_TX = "Insert New Customer";
  static final String INSERT_ITEM_TX = "Insert Order Item";
  static final String INSERT_ORDER_TX = "Insert Order";
  static final String UPDATE_ORDER_TX = "Update Order";
```

```
  static final String UPDATE_WAREHOUSE_TX = "Update Warehouse";
  static final String GET_CUSTOMER_DETAILS_TX = "Get Customer
Details";
  static final String BROWSE_PENDING_ORDERS = "Browse Pending
Orders";
  static final String BROWSE_BY_PROD_ID = "Browse Product by ID";
  static final String BROWSE_BY_CATEGORY_TX = "Browse Products by
Category";
  static final String BROWSE_BY_CAT_QUAN_TX = "Browse Products by
Quantity";
  static final String BROWSE_BY_ORDER_ID = "Browse Orders by ID";
  static final String BROWSE_ORDER_DETAILS = "Browse Order Details";
  static final String UPDATE_PENDING_ORDERS = "Update Pending
Orders";
  static final String GET_ORDER_BY_CUSTOMER_TX = "Browse Order by
Customer";
  static final int MIN_CATEGORY = 1;
  static final int MAX_CATEGORY = 10;
  static final int MAX_BROWSE_CATEGORY = 6;
  static final int MAX_CREDITLIMIT = 5000;
  static final int MIN_CREDITLIMIT = 100;
  static final int MIN_SALESID = 145;
  static final int MAX_SALESID = 171;
  static final int MIN_PRODS_TO_BUY = 2;
  static final int MAX_PRODS_TO_BUY = 6;
  static final int MIN_PROD_ID = 1;
  static final int MAX_PROD_ID = 288;
  static int MIN_WAREHOUSE_ID = 1;
  static int MAX_WAREHOUSE_ID = 9;
  static final int MIN_ORDER_ID = 1;
  static final int MAX_ORDER_ID = 146610;
  static final int AWAITING_PROCESSING = 4;
  static final int ORDER_PROCESSED = 10;
  static int MIN_CUSTID = 101;
  static int MAX_CUSTID = 500000;
  private PreparedStatement catPs = null;
  private PreparedStatement catqPs = null;
  private PreparedStatement custPs = null;
  private PreparedStatement insLogon = null;
  private PreparedStatement orderPs = null;
  private PreparedStatement orderPs2 = null;
  private PreparedStatement prodPs = null;
  private PreparedStatement prodqPs = null;

  public void logon(Connection connection, int custid) throws
SQLException {
    Date currentTime = new Date(System.currentTimeMillis());
    PreparedStatement insLogon = connection.prepareStatement("insert
into logon values(?,?)");
    insLogon.setInt(1, custid);
    insLogon.setDate(2, currentTime);
    insLogon.executeUpdate();
    connection.commit();
    insLogon.close();
  }

  public void getMaxandMinCustID(Connection connection) throws
SQLException {
```

```
    PreparedStatement mmPs = null;
    ResultSet rs = null;
    try {
      mmPs = connection.prepareStatement("select min(customer_id),
max(customer_id) from customers");
      rs = mmPs.executeQuery();
      if (rs.next()) {
        MIN_CUSTID = rs.getInt(1);
        MAX_CUSTID = rs.getInt(2);
      }
    } finally {
      rs.close();
      mmPs.close();
    }

    //should be called only once;
  }

  public void getMaxandMinWarehouseID(Connection connection) throws
SQLException {
    PreparedStatement mmPs = null;
    ResultSet rs = null;
    try {
      mmPs = connection.prepareStatement("select min(warehouse_id),
max(warehouse_id) from warehouses");
      rs = mmPs.executeQuery();
      if (rs.next()) {
        MIN_WAREHOUSE_ID = rs.getInt(1);
        MAX_WAREHOUSE_ID = rs.getInt(2);
      }
    } finally {
      rs.close();
      mmPs.close();
    }

  }

  public void insertAdditionalWarehouses(Connection connection, int
numWarehouses) throws SQLException {
    PreparedStatement iaw = null;
    try {
      iaw = connection.prepareStatement("insert into
warehouses(warehouse_id, warehouse_name, location_id) values
(?,?,?)");

      for (int i = 0; i < numWarehouses; i++) {
        int whid = (MAX_WAREHOUSE_ID + i) + 1;
        iaw.setInt(1, whid);
        iaw.setString(2, "Warehouse Number " + whid);
        iaw.setInt(3, RandomUtilities.randomInteger(1, 9999));
        iaw.executeUpdate();
      }
      connection.commit();

    } finally {
      iaw.close();
    }
  }
```

```
  public void getCustomerDetails(Connection connection, int custId)
throws SQLException {
    ResultSet rs = null;
    try {
      custPs = connection.prepareStatement("select customer_id,
cust_first_name ,cust_last_name ,nls_language ,nls_territory
,credit_limit ,cust_email ,account_mgr_id  from customers where
customer_id = ?");
      custPs.setInt(1, custId);

      rs = custPs.executeQuery();
      rs.next();
    } finally {
      hardClose(rs);
      hardClose(custPs);
    }

  }

  public double getProductDetailsByID(Connection connection, int
prodID) throws SQLException {
    ResultSet rs = null;
    double price = 0;

    try {
      prodPs = connection.prepareStatement(" select product_id,
product_name, product_description, category_id, weight_class,
supplier_id, product_status, list_price, min_price, catalog_url from
product_information where product_id = ?");
      prodPs.setInt(1, prodID);
      rs = prodPs.executeQuery();
      while (rs.next()) {
        price = rs.getDouble(8);
      }
    } catch (SQLException se) {
      throw se;
    } finally {
      hardClose(rs);
      hardClose(prodPs);
    }

    return price;
  }

  public List<Integer> getProductDetailsByCategory(Connection
connection, int catID) throws SQLException {
    List<Integer> result = new ArrayList<Integer>();
    ResultSet rs = null;
    try {
      catPs = connection.prepareStatement(" select product_id,
product_name, product_description, category_id, weight_class,
supplier_id,  product_status  list_price, min_price, catalog_url from
product_information where category_id = ?");
      catPs.setInt(1, catID);

      rs = catPs.executeQuery();
      while (rs.next()) {
```

```
        result.add(rs.getInt(1));
      }
    } finally {
      hardClose(rs);
      hardClose(catPs);
    }
    return result;
  }

  public int getProductQuantityByID(Connection connection, int ID)
throws SQLException {
    int quantity = 0;
    ResultSet rs = null;
    try {
      prodqPs = connection.prepareStatement(" select  p.product_id,
product_name, product_description, category_id, weight_class,
supplier_id,  product_status,  list_price, min_price, catalog_url,
quantity_on_hand, warehouse_id from  product_information p,
inventories i where   i.product_id = ?  and   i.product_id =
p.product_id");
      prodqPs.setInt(1, ID);

      rs = prodqPs.executeQuery();
      if (rs.next()) {
        quantity = rs.getInt(11);
      }
    } finally {
      hardClose(rs);
      hardClose(prodqPs);
    }
    return quantity;
  }

  public void getProductQuantityByCategory(Connection connection, int
catID) throws SQLException {
    ResultSet rs = null;
    try {
      catqPs = connection.prepareStatement(" select /*+ first_rows */
p.product_id, product_name, product_description, category_id,
weight_class, supplier_id, product_status, list_price, min_price,
catalog_url,  quantity_on_hand, warehouse_id from
product_information p,  inventories i where  category_id = ? and
i.product_id = p.product_id");
      catqPs.setInt(1, catID);

      rs = catqPs.executeQuery();
      rs.next();
    } finally {
      hardClose(rs);
      hardClose(catqPs);
    }

  }

  public void getOrderByID(Connection connection, int orderID) throws
SQLException {
    ResultSet rs = null;
    try {
```

```
      orderPs = connection.prepareStatement(" select /*+ first_rows
*/    order_id,   order_date,   order_mode,   customer_id,
order_status,   order_total,   sales_rep_id,   promotion_id from
orders where    order_id = ?");
      orderPs.setInt(1, orderID);

      rs = orderPs.executeQuery();
      rs.next();
    } finally {
      hardClose(rs);
      hardClose(orderPs);
    }

  }

  public void getOrderDetailsByOrderID(Connection connection, int
orderID) throws SQLException {
    ResultSet rs = null;
    try {
      orderPs2 = connection.prepareStatement(" SELECT /*+ use_nl */
o.order_id,   line_item_id,   product_id,   unit_price,   quantity,
order_mode,   order_status,   order_total,   sales_rep_id,
promotion_id,   c.customer_id,   cust_first_name,   cust_last_name,
credit_limit,   cust_email  FROM    orders o ,   order_items oi,
customers c WHERE    o.order_id = oi.order_id  and   o.customer_id =
c.customer_id  and   o.order_id = ?");
      orderPs2.setInt(1, orderID);

      rs = orderPs2.executeQuery();
      rs.next();
    } finally {
      hardClose(rs);
      hardClose(orderPs2);
    }

  }

  /*public void setMaxSleepTime(long newMaxSleepTime) {
    maxSleepTime = newMaxSleepTime;
  }

  public void setMinSleepTime(long newMinSleepTime) {
    minSleepTime = newMinSleepTime;
  }*/

}
```

Programa ProcessOrders.java

```
package com.dom.benchmarking.swingbench.transactions;

import com.dom.benchmarking.swingbench.event.JdbcTaskEvent;
import com.dom.benchmarking.swingbench.kernel.SwingBenchException;
import com.dom.benchmarking.swingbench.kernel.SwingBenchTask;
import com.dom.benchmarking.swingbench.utilities.RandomGenerator;
```

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;


import java.util.Map;


public class ProcessOrders extends OrderEntryProcess {

  private PreparedStatement orderPs3 = null;
  private PreparedStatement updoPs = null;
  private int orderID;

  public ProcessOrders() {
  }

  public void close() {
  }

  public void init(Map parameters) {
  }

  public void execute(Map params) throws SwingBenchException {
    Connection connection =
(Connection)params.get(SwingBenchTask.JDBC_CONNECTION);
    initJdbcTask();
    ResultSet rs = null;

    long executeStart = System.nanoTime();

    try {
      try {
        orderPs3 = connection.prepareStatement("SELECT /*+
first_rows index(customers, customers_pk) index(orders,
order_status_ix) */  o.order_id, line_item_id, product_id,
unit_price, quantity, order_mode, order_status, order_total,
sales_rep_id, promotion_id, c.customer_id, cust_first_name,
cust_last_name, credit_limit, cust_email, order_date FROM orders o ,
order_items oi, customers c WHERE o.order_id = oi.order_id  and
o.customer_id = c.customer_id  and o.order_status <= 4");

        rs = orderPs3.executeQuery();
        rs.next();
        orderID = rs.getInt(1);
        addSelectStatements(1);
        thinkSleep(); //update the order
        updoPs = connection.prepareStatement("update /*+
index(orders, order_pk) */ " + "orders " + "set order_status = ? " +
"where order_id = ?");
        updoPs.setInt(1,
RandomGenerator.randomInteger(AWAITING_PROCESSING + 1,
ORDER_PROCESSED));
        updoPs.setInt(2, orderID);
        updoPs.execute();
        addUpdateStatements(1);
```

```
        connection.commit();
        addCommitStatements(1);
      } catch (SQLException se) {
        throw new SwingBenchException(se.getMessage());
      } finally {
        hardClose(rs);
        hardClose(orderPs3);
        hardClose(updoPs);
      }

      processTransactionEvent(new JdbcTaskEvent(this, getId(),
(System.nanoTime() - executeStart), true, getInfoArray()));
    } catch (SwingBenchException sbe) {
      processTransactionEvent(new JdbcTaskEvent(this, getId(),
(System.nanoTime() - executeStart), false, getInfoArray()));

      try {
        connection.rollback();
      } catch (SQLException er) {
      }

      throw new SwingBenchException(sbe.getMessage());
    }
  }

}
```

GLOSSÁRIO

GLOSSÁRIO

AMD - *Advanced Micro Devices.* Empresa fabricante de circuitos integrados, especialmente processadores, também chamados de UCP (unidade central de processamento).

Backup - Refere-se a uma cópia de um recurso que pode ser restaurada ou ativada em caso de falha do recurso original.

Bare metal - É o *hardware* puro, sem nenhum sistema operacional instalado.

Benchmark - É um programa, ou conjunto de programas, que podem ser executados em diferentes servidores para dar uma medida aproximada de seus desempenhos. Apesar de um *benchmark* não caracterizar completamente o desempenho total de um sistema, seus resultados podem ser usados como referência para o desempenho real esperado.

Bit - *Binary Digit* – Dígito binário. É a menor unidade de informação que pode ser armazenada ou transmitida.

Cisco - Empresa multinacional americana fabricante de soluções para redes e comunicações, com destaque para a fabricação de roteadores e *swiches*.

Centro de Dados - em inglês *Data Center.* É o centro de processamento de dados até recentemente chamado de CPD (Centro de Processamento de Dados).

Drivers - módulos de código específicos para acessar os dispositivos físicos. Existe um *driver* para cada tipo de dispositivo, como discos rígidos, portas *USB*, placas de vídeo, dentre outros. Muitas vezes o *driver* é construído pelo próprio fabricante do *hardware* e fornecido em forma compilada (em linguagem de máquina) para ser acoplado ao restante do SO. Outras vezes ficam no SO.

EMC - Empresa multinacional norte-americana que fornece sistemas para infraestrutura de informação, software e serviços. Seu principal produto é o *Symmetrix* que é a base para redes de armazenamento de dados de vários centros de dados. Uma das empresas do Grupo *EMC* é a *VMware.*

Emulex - Empresa americana fabricante de soluções para infraestrutura de rede de armazenamento (em inglês SAN - *storage area network*)

E/S - Entrada/Saída. Termo utilizado no ramo da computação, indicando entrada (inserção) de dado por meio de algum código ou programa, para algum outro programa ou *hardware*, bem como a sua saída (retorno de dados)

como resultado de alguma operação de algum programa, consequentemente resultado de alguma entrada.São exemplos de unidades de entrada de um computador: teclado, mouse, leitor de código de barras. São exemplos de unidades de saída de um computador: monitor, impressora, disco rígido.

Firewall - Dispositivo de rede uma que tem por objetivo aplicar uma política de segurança a um determinado ponto de controle da rede. Sua função consiste em regular o tráfego de dados entre redes distintas e impedir a transmissão e/ou recepção de acessos nocivos ou não autorizados de uma rede para outra.

Firmware - Conjunto de instruções operacionais programadas diretamente no *hardware* de um equipamento eletrônico.

Framework - Conjunto de conceitos usado para resolver um problema de domínio específico.

Gb/s - *Gigabits* por segundo. Unidade de armazenamento de informações ou dados. Um *gigabit* é igual a um bilhão de bits.

Gigabytes - Unidade de armazenamento de informações ou dados. Um *gigabyte* é igual a oito bilhões de bits.

Hardware - Conjunto de componentes eletrônicos, circuitos integrados e placas que comunicam entre si através de barramentos e que constituem a parte física do computador.

Hypervisor - monitor de máquinas virtuais que é instalado e executado diretamente no hardware da máquina física.

Intel - *Intel Corporation* é a contração de *Integrated Electronics Corporation,* empresa multinacional de origem americana fabricante de circuitos integrados, especialmente microprocessadores. Empresa criadora da série *x86* de processadores.

JDBC - *Java Database Conectivity*. É uma Interface Aplicativa de Programação para acesso SQL a banco de dados por programas Java.

Kernel - Componente central da maioria dos computadores que serve de ponte entre as aplicações e o processamento real de dados feito a nível de hardware. As responsabilidades do *kernel* incluem gerenciar a comunicação entre componentes de *hardware* e *software.*

Mainframe - Computador de grande porte fabricado pela *IBM*

Mb/s - *Megabits* por segundo. Unidade de armazenamento de informações ou dados. Um *megabit* é igual a um milhão de bits.

MB/s - *Megabytes* por segundo. Unidade de armazenamento de informações ou dados. Um *megabyte* é igual a oito milhões de bits.

MMV - Monitor de Máquina Virtual.

MV - Máquina Virtual.

OLAP - *On-Line Analytical Processing*. Processamento analítico em tempo real.

OLTP - *On-Line Transation Processing*. Processamento de transações em tempo real.

Processador - Também chamado de UCP (Unidade Central de Processamento). É a parte de um sistema de computador que executa as instruções de um programa. É o elemento primordial na execução das funções de um computador.

RAM - *Random Access Memory*. Memória de acesso aleatório que permite a leitura e gravação. É utilizada como memória primária em sistemas eletrônicos digitais.

RPM - Rotações por minutos. É uma unidade de freqüência angular ou velocidade angular utilizada para informar a quantidade de rotações por segundo que será utilizada para ler os setores do disco.

Root mode - Modo raiz. Termo utilizado para descrever o estado privilegiado em um contexto de execução de instruções em um sistema operacional.

SAS - *Serial Attached SCSI*. Protocolo de gerenciamento e armazenamento de dados mais confiável, rápido e versátil que seu antecessor, o SCSI.

SGBD - Sistema de Gestão de Base de Dados. Um Sistema de Gestão de Bases de Dados é um conjunto de programas que permitem armazenar, modificar e extrair informação de um banco de dados.

SO - Sistema Operacional.

Software - Conjunto de instruções e dados processado pelos circuitos eletrônicos do *hardware*. É a camada, colocada sobre o *hardware*, que transforma o computador em algo útil para o ser humano.

Storage - subsistema para armazenamento de dados em discos rígidos externos a um servidor.

Switch - Dispositivo utilizado em redes de computadores para reencaminhar frames entre os diversos nós de uma rede.

Terabytes - Unidade de armazenamento de informações ou dados. Um *terabyte* é igual a oito trilhões de bits.

Segmento - Linha de execução, em inglês *thread*. É uma forma de um processo dividir a si mesmo em duas ou mais tarefas que podem ser executadas simultaneamente. O suporte a segmentos é fornecido pelo sistema operacional no nível de *kernel*.

TI - Tecnologia da Informação

TPC-C - *Benchmark* para processamento de transações em tempo real (*OLTP*). É utilizado para comparar o desempenho em configurações de *hardware* e *software.*

UCP - Também chamado de processador. É a parte de um sistema de computador que executa as instruções de um programa. É o elemento primordial na execução das funções de um computador.

USB - *Universal Serial Bus.* Tipo de conexão plug and play que permite a conexão de periféricos sem a necessidade de desligar o computador.

VMware - *Software* que permite a instalação e utilização de um sistema operacional dentro de outro dando suporte real a *software* de outros sistemas operativos. É fabricado pela empresa de mesmo nome (*VMware*) que é uma empresa dentro do Grupo *EMC*.