DENIS RYOJI OGURA

UMA METODOLOGIA PARA CARACTERIZAÇÃO DE APLICAÇÕES EM AMBIENTES DE COMPUTAÇÃO NAS NUVENS

São Paulo
2011

DENIS RYOJI OGURA

UMA METODOLOGIA PARA CARACTERIZAÇÃO DE APLICAÇÕES EM AMBIENTES DE COMPUTAÇÃO NAS NUVENS

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Mestre em Engenharia Elétrica

São Paulo
2011

DENIS RYOJI OGURA

UMA METODOLOGIA PARA CARACTERIZAÇÃO DE APLICAÇÕES EM AMBIENTES DE COMPUTAÇÃO NAS NUVENS

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Mestre em Engenharia Elétrica

Área de Concentração:
Sistemas Digitais

Orientador: Prof. Doutor
Edson Toshimi Midorikawa

São Paulo
2011

DEDICATÓRIA

Dedico esta dissertação aos meus filhos Renan Ogura e Nicolly Ogura, que muitas vezes tiveram de entender a minha ausência, por precisar me dedicar ao desenvolvimento desta dissertação. Também gostaria de dedicar a dissertação à minha mãe Mary Ogura e ao meu saudoso pai Yoshisuke Ogura.

AGRADECIMENTOS

Aos meus filhos Renan e Nicolly, à minha avó Wilma, aos meus irmãos Alberto, Paulo Tiago e Matheus, que me suportaram durante o mestrado.

Ao meu saudoso pai, Yoshisuke Ogura, que me incentivou a ingressar na vida acadêmica, assim como à minha mãe, Mary Ogura, que também me ajudou incondicionalmente.

Ao professor orientador Edson Toshimi Midorikawa pelos três anos de convivência e ensinamentos, e pela amizade e paciência nas horas boas e nas difíceis.

À professora Líria Matsumoto Sato pelas diversas ajudas, diretas e indiretas.

A todos os professores da Escola Politécnica da Universidade de São Paulo pelas matérias ministradas durante meu curso de mestrado.

Aos professores Edson Midorikawa, Líria Sato e Pedro Corrêa pelos comentários e sugestões efetuadas na qualificação dessa dissertação.

Aos meus amigos Artur, Charles, Kakugawa e Piantola, que me ajudaram em diversas situações.

A todos os meus amigos do laboratório LAHPC.

Às empresas HP e IBM, que viabilizaram minha participação no processo de mestrado.

RESUMO

Computação nas nuvens é um novo termo criado para expressar uma tendência tecnológica recente que virtualiza o data center. Esse conceito busca um melhor aproveitamento dos recursos computacionais e dos aplicativos corporativos, virtualizados por meio de programas de virtualização de sistema operacional (SO), plataformas, infraestruturas, softwares, entre outros.

Essa virtualização ocorre por intermédio de máquinas virtuais (MV) para executar aplicativos nesse ambiente virtualizado. Contudo, uma MV pode ser configurada de tal forma que seu desempenho poderá ter um atraso no processamento por conta de gargalo(s) em algum hardware alocado.

A fim de maximizar a alocação do hardware na criação da MV, foi desenvolvido um método de caracterização de aplicações para a coleta de dados de desempenho e busca da melhor configuração de MV. A partir desse estudo, pode-se identificar pelo *workload* a classificação do tipo de aplicação e apresentar o ambiente mais adequado, um recomendado e não recomendado. Dessa forma, a tendência de se obter um desempenho satisfatório nos ambientes virtualizados pode ser descoberta pela caracterização dos programas, o que possibilita avaliar o comportamento de cada cenário e identificar situações importantes para seu bom funcionamento. Para provar essa linha de raciocínio, foram executados programas mono e multiprocessador em ambientes de monitores de máquinas virtuais. Os resultados obtidos foram satisfatórios e estão de acordo com cada característica de aplicação conhecida previamente. Porém, podem ocorrer situações de exceção nesse método, principalmente quando o monitor de máquinas virtuais, é submetido a processamentos intensos. Com isso, a aplicação pode ter um atraso no processamento por conta do gargalo de processamento no monitor de máquinas virtuais, o que modifica o ambiente ideal dessa aplicação. Portanto, este estudo apresenta um método para identificar a configuração ideal para a execução de um aplicativo.

Palavras-chave: Computação nas nuvens, aplicações nas nuvens, caracterização de sistemas, monitores de máquinas virtuais, máquinas virtuais.

ABSTRACT

Cloud computing represents a new age, raised to express a new technology trending that virtualizes the data center. This concept advanced to make a better use of the computational resources and corporate application, virtualizing through the programs of operating systems virtualization, platform, infrastructure, software, etc.

This virtualization occurs through the virtual machine (VM) to execute virtualized applications in this environment. However, a VM may be configured in such a way that the performance delays on processing, due to overhead or other hardware allocation itself.

In order to maximize the hardware allocation on MV creation, it was developed a methodology of application characterization to collect performance data and achieve the best VM configuration. After this study, based on workload metric, it is possible to identify the classification of the application type and present the best configuration, the recommended environment and the not recommended. This way, the trend to achieve a satisfactory performance in virtualized environment may be discovered through the program characterization, which possibly evaluate the behavior of each scenario and identify important conditions for its proper operation. In order to prove this argument, mono and multi core applications under monitors of virtual machines were executed. The collected results were satisfactory and are aligned with each previously known application characteristic. However, it may occur exceptions in this method, mainly when the monitor of the virtual machine monitor is submitted with high volume of processing.

Keywords: Cloud Computing, cloud applications, system characterization, virtual machines, virtual machine monitors, virtual machines.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

GLOSSÁRIO

API	*Application Program Interfaces*
Benchmark	Ponto de referência de uma medida
BPS	*Bits Per Second*
BT	*Block Tridiagonal*
CAPEX	*Capital Expenses*
CG	*Conjugate Gradient*
CPU	*Central Processor Unit*
DOM0	*Domínio 0*
EP	*Embarrassingly Parallel*
ERP	*Enterprise Resource Planning*
FS	*File System*
FT	*Fourier Transform*
HD	*Hard Disk*
HPC	High Performance Computing
IaaS	*Infrastrucuture As A Service*
IO	Input/Output (entrada e saída)
IS	*Integer Sort*
ISP	Internet Service Providers
KVM	*Kernel Virtual Machine*
LU	*Lower and Upper*
MFLOPS	*Millions of Floating-point Operations Per Second*
MG	*Multigrid calculation*
MIPS	*Millions Instructions Per Second*
MPI	Message Passing Interface
MTBE	*Mean Time Between Error*
MTBF	*Mean Time Between Fail*
MTTF	*Mean Time to Fail*
MTTR	*Mean Time to Repair*
MV	Máquina Virtual
NFS	*Network File System*

NFS	*Network File System*
NPB	*NAS Parallel Benchmark*
OPEX	*Operating Expenses*
PaaS	*Platform As A Service*
QOS	*Quality of Services*
RTT	*Round Trip Time*
SaaS	*Software As A Service*
SGBD	Sistema Gerenciador de Banco de Dados
SLA	*Service Level Agreement*
SO	Sistemas Operacionais
SP	*Scalar and Pentadiagonal*
SQL	Structure Query Language
SSD	Solid State Drive
TI	Tecnologia da Informação
TPC	*Transactional Processing Performance Council*
TPS	*Transactions Per Second*
VCPU	*Virtual Central Processor Unit*
VMM	*Virtual Machine Monitor*
VPN	Virtual Private Network

1. INTRODUÇÃO

Atualmente, o conceito de computação nas nuvens tem sido alvo de muitas discussões, pois cada provedor de serviço implementa computação nas nuvens de acordo com sua própria política, e também pelo fato de não existir um padrão.

Mesmo não havendo um padrão, a computação nas nuvens é descrita basicamente em três arquiteturas: *infrastrucutre as a service* (IaaS), *plataform as a service* (PaaS), *software as a service* (SaaS). Com essas estruturas básicas, o provedor organiza o ambiente computacional provendo para o cliente uma máquina virtual (MV) simples, uma interface com uma plataforma (Google App Engine, Microsoft Windows Azure, Amazon S3, Force.com, Morph, Bungee Connect) ou o produto ser disponibilizado diretamente para o cliente.

Na perspectiva do provedor de infraestrutura na nuvem, é vantajosa a mistura de diferentes equipamentos ou arquiteturas de computadores para a execução de um determinado programa. Para isso, se a aplicação possuir suporte a processamento distribuído e paralelo, as vantagens aumentarão de acordo com a possibilidade de se utilizar mais de uma máquina ou um multiprocessador para um aplicativo. Com isso, o provedor de nuvem tem a possibilidade de criar soluções de escalabilidade e flexibilidade no gerenciamento de máquinas virtuais (MV) de uma forma simples, conforme a necessidade do cliente da nuvem.

Ao fazer uso desta flexibilidade de alocação dos recursos computacionais conforme a demanda, o provedor aproveita ao máximo os equipamentos disponíveis em seu *data center* para o processamento das aplicações na nuvem.

A tecnologia de computação nas nuvens ainda tem algumas dificuldades, principalmente em termos de desempenho, confidencialidade de dados, escalabilidade, segurança e armazenamento de dados (ARMBRUST, 2009). O provedor da nuvem precisa modificar os mecanismos de provisionamento de custos, desempenho e segurança de dados. Além disso, deve fornecer recursos ou meios para garantir integridade de dados, confidencialidade e auditoria nos ambientes de máquinas virtuais. Esses requisitos não funcionais são importantes para o controle e a garantia de execução de uma aplicação comercial. Além do mais, é importante que haja um

contrato de prestação de serviço (*Service Level Agreement - SLA*) (PATEL, 2009) com as regras e políticas que regem o funcionamento das atividades desse provedor de nuvem. O SLA garante, para ambas as partes, qualidade e disponibilidade do ambiente e inclui o que está ou não no escopo do ambiente virtual, entre outros aspectos. Caso o provedor de nuvem necessite utilizar ambientes computacionais de empresas terceiras, é preciso que seja incluído no contrato, para que o cliente tenha o conhecimento dessa possibilidade de transferência da máquina virtual para outra infraestrutura. Uma vez firmado esse contrato, os clientes podem usufruir da máquina virtual conforme suas necessidades, e utilizar desse ambiente o tempo que for necessário. Pode-se, apenas, ter métodos para monitoração de desempenho, algoritmo de tolerância à falha no monitor de máquinas virtuais, entre outros métodos para a administração do ambiente computacional.

Na perspectiva do cliente da nuvem, ocorre uma mudança de cultura empresarial em termos de projeto e arquitetura das soluções computacionais. O cliente pode configurar os aplicativos corporativos em formato de MVs em sua infraestrutura privada, e, na falta de recurso para o processamento, transferir essa MV para a nuvem, flexibilizando a arquitetura das aplicações, com a possibilidade de escalar diferentes infraestruturas e combinar ambientes privados e públicos. Outra solução interessante da nuvem é a possibilidade de criar um ambiente redundante; caso o sistema falhe na infraestrutura privada, é ativada a MV de contingência na nuvem pública, contendo os mesmos aplicativos, bibliotecas, *framework*, e todos os recursos para executar os serviços necessários.

Com relação ao provedor da nuvem, também se está passando por uma mudança empresarial, pois ele precisa contabilizar a utilização das MVs do cliente da nuvem por hora, que é convertida em tarifas para o cliente. Conceitualmente, denomina-se *pay-as-you-go* (paga-se pelas horas utilizadas) (PATEL, 2009) ou *operating expenses* (Opex – despesas operacionais) (ARMBRUST, 2009).

O modelo Opex consiste na forma como uma empresa investe em infraestrutura, contratando serviços – em infraestrutura de terceiros – para operar aplicativos sem a aquisição de recursos, equipamentos, programas etc. Um outro modelo é o *capital*

expenses (Capex – despesas de capital) (ARMBRUST, 2009), que propõe o investimento na aquisição de equipamentos, programas para a operação do sistema e ferramentas ou componentes externos para complementar o projeto. A empresa cliente da nuvem não precisa adquirir ou administrar o dimensionamento dos equipamentos computacionais para um determinado *workload* de uma aplicação, porque, com a MV no provedor da nuvem, toda a responsabilidade de manter processamento, monitoração e dimensionamento da capacidade de processamento pertence ao provedor da nuvem pública. Por outro lado, o provedor da nuvem aproveita os equipamentos computacionais em seu *data center* que se encontram em operação para um objetivo (sistema interno da empresa, por exemplo), e compartilha o recurso com os clientes da nuvem, fazendo o mesmo equipamento ser utilizado com objetivos diferentes.

Uma vez criada a MV, as aplicações podem ser transferidas e configuradas no ambiente da nuvem pública, necessitando haver uma interação entre o cliente e a MV. Pelo fato dessa aplicação não ser conhecida pelo provedor da nuvem, atualmente se cria apenas a MV solicitada e se disponibiliza esse ambiente para o cliente por meio de uma interface *web* ou outro meio para acessar a MV. Se o provedor da nuvem tiver um método pelo qual essa aplicação possa ser avaliada previamente, ele poderá criar um ambiente de MVs que comporte tal aplicação, mas com um desempenho adequado de acordo com a característica do programa. Para que esse método possa ser viabilizado, é preciso um estudo mais profundo sobre o desempenho das aplicações em ambientes virtuais.

Esta dissertação apresenta uma proposta de método para caracterizar uma aplicação por meio da coleta de dados de desempenho (instrumentação), classificando-os em taxonomias de aplicativos. Cada grupo de aplicação terá uma configuração ideal ou indicada para esse programa determinado e comparada a outros aplicativos já instrumentados. A partir deste estudo, espera-se identificar um método que seja aplicado nos *data centers* dos provedores de nuvem, por meio dos quais se pode dimensionar qual equipamento – ou conjuntos de equipamentos – é adequado para processar a aplicação do cliente da nuvem. Então, essa dissertação apresenta uma proposta de metodologia de identificação de características de aplicações com base em

processamento, memória e *input-output* (I/O). Essa instrumentação traz indicativos dos recursos de que essa aplicação necessita. O provedor da nuvem pode aproveitar esse método para criar estruturas com base no inventário de computadores de diferentes marcas, arquitetura e SO, para estruturar alocação de uma forma concisa e coerente. Adicionalmente, o provedor da nuvem pode criar domínios de equipamentos para processar cada grupo de taxonomia definido no método. Uma vez conhecedor do ambiente, o provedor da nuvem pode aumentar ou diminuir os recursos alocados para a nuvem pública e, quando preciso, pode simplesmente retirá-;ps do domínio da nuvem e alocar onde for necessário, ou apenas utilizar a própria nuvem pública para fazer os processamentos da corporação.

1.1 OBJETIVO

O objetivo desta dissertação é definir uma metodologia para a caracterização de aplicações em ambientes de computação nas nuvens. A partir deste método, é possível identificar características da aplicação que possam ser classificadas por taxonomia de programas, sendo que cada categoria tem uma solução de arquitetura indicada ou recomendada. Esta metodologia propõe uma alternativa para que os provedores da nuvem possam controlar, configurar e escalar os ambientes computacionais conforme a necessidade do cliente ou conforme a necessidade de negócio que o provedor da nuvem possa vir a ter.

1.2 METODOLOGIA

Esta dissertação foi desenvolvida em duas etapas. Na primeira, foi feito um estudo inicial com o intuito de identificar características de consumo de CPU, disco e a influência do monitor de máquinas virtuais em aplicações distribuídas, paralelas e transacionais. Na segunda parte, foi desenvolvida uma metodologia para caracterização de aplicação, e nela foram definidas três fases. Na primeira fase, foi criada uma MV contendo todos os aplicativos do experimento e os comandos de instrumentação, para que estes últimos possam ser executados paralelamente, coletando os dados de

consumo de cada métrica. Na segunda etapa, foram importadas os valores da instrumentação com o cálculo da média aritmética de cada métrica coletada de cada arquivo de saída da instrumentação da primeira fase. E, por fim, a terceira fase é a comprovação de que a configuração de MV sugerida obteve um comportamento de desempenho esperado. Os *scripts* de processamento do programa e da instrumentação foram desenvolvidos em *Bash* para que pudessem ser feitos processamentos em conjunto e diversas iterações do mesmo aplicativo.

Na instrumentação, foram coletados dados de I/O de *file system*, medindo o consumo de memória para o processo e a utilização de CPU. Após a análise dos resultados, identificou-se que o monitor de máquinas virtuais e o NFS remoto obtiveram desempenho abaixo do esperado e que seria necessária uma investigação com detalhes para saber o motivo do atraso. Estudou-se, então, a hipótese de o dom0 estar sobrecarregado. Para uma análise com detalhes, foi preciso coletar mais dados no dom0. Assim, foram executados o `iostat` e o `xentop` coletando dados de I/O e a utilização de CPU do monitor de máquinas virtuais; no cenário com o NFS remoto, foi executado adicionalmente o comando `iostat` no servidor de NFS remoto.

1.3 CONTRIBUIÇÃO DESTE TRABALHO

A contribuição desta dissertação consiste em apresentar uma metodologia de categorização de programas computacionais para ambientes de computação em nuvens, que, por intermédio dessa tecnologia, pode classificar uma aplicação qualquer em grupos. Para cada categoria, há uma solução tecnológica associada à configuração da MV, para se atingir um desempenho satisfatório de acordo com a característica da aplicação. O método permite incrementar o conhecimento das aplicações conforme outras aplicações forem sendo instrumentadas e incluídas na metodologia, pois a classificação é armazenada em um banco de dados relacional de forma organizada e estruturada. Pelo fato de essa leitura estar armazenada em uma estrutura de um sistema gerenciador de banco de dados (SGBD), ela permite correlação e análise de outras leituras em comparação com a aplicação e, por via da classificação *Fuzzy*, viabiliza a divisão das taxonomias em três grupos (Tabela 23): utilização intensa do

recurso (+), média e baixa (-).

Nos experimentos, podem-se coletar dados das aplicações e analisá-los em tabelas e gráficos. Os resultados estavam de acordo com a expectativa inicial das execuções dos cenários. Contudo, uma das aplicações teve um comportamento inesperado: foi processado em um tempo abaixo da expectativa. Esse desvio foi investigado e foram apresentados dados e fatos que justificam essa ineficiência da aplicação (Capítulo 4).

1.4 ESTRUTURA DOS CAPÍTULOS

Esta dissertação está organizada da seguinte forma. O Capítulo 2 apresenta os conceitos necessários para a metodologia de caracterização das aplicações e indica as soluções utilizadas: computação nas nuvens (IaaS, PaaS, SaaS), com exemplos de implementações para cada estrutura. O dito capítulo desenvolve também outros assuntos, como virtualização, método de instrumentação, pacote sysstat e comando pmap.

O Capítulo 3 explora a proposta de metodologia desta dissertação. Para isso, são apresentados os seguintes pontos: a forma como as aplicações são caracterizadas, um primeiro estudo sobre a caracterização de aplicações, resultados do estudo inicial, a metodologia da caracterização de aplicações segmentada em três fases, e por fim, os trabalhos relacionados.

Já o Capítulo 4 aborda os resultados e análises dos experimentos por intermédio das aplicações `pbzip2` e `lame` executadas em MVs, para verificar o comportamento destas aplicações com o processo de instrumentação sendo executado em paralelo. Depois disso, é feita uma análise dos valores coletados e apresentados em formato de gráficos. Por fim, são apresentadas as considerações finais dos experimentos da metodologia.

O Capítulo 5 apresenta as conclusões desta dissertação, assim como as contribuições alcançadas com esta dissertação, sugestões de trabalhos futuros.

E por fim, os Apêndices com os códigos *scripts* utilizados no processo de execução da metodologia proposta nesta dissertação.

2. CONCEITOS BÁSICOS

Neste capítulo, são abordados conceitos requeridos para a conceituação e o entendimento da proposta da metodologia de instrumentação de aplicações em um ambiente de computação nas nuvens. Serão estudados conceitos importantes como computação nas nuvens, virtualização, instrumentação e caracterização de aplicações.

2.1 COMPUTAÇÃO NAS NUVENS

Aparentemente, computação nas nuvens é um termo atual. Contudo, a origem do conceito de virtualização dos componentes e recursos de TI, desde a década de 1960, vem sendo apresentada por John McCarthy (PARKHILL, 1966) como computação utilitária (*utility computing*). A computação utilitária foi conceituada como um pacote de recursos computacionais, de armazenamento de dados e de serviços. Com esse modelo de computação, pode-se fazer uma analogia com empresas de utilidades públicas, como fornecimento de eletricidade, água, gás natural, telefone, internet etc. Todos esses serviços são fornecidos pelas concessionárias prestadores de serviços, apenas se paga pelo consumo correspondente ao período medido. A computação utilitária teve uma conceituação no sentido de aluguel de parte dos equipamentos, particionando-os em frações de modo que pudessem ser compartilhados com outras empresas. Sendo assim, os provedores de serviço adquiriam os equipamentos, conforme a necessidade e a demanda por mais recursos computacionais. Portanto, a empresa que consome o serviço do provedor tem um ganho em termos de implementação e configuração do ambiente computacional, porque não precisa adquirir equipamentos, contratar suporte técnico, definir políticas de segurança etc. Uma vez que esse ambiente alugado esteja sob responsabilidade do provedor de serviço, todas essas necessidades passam a fazer parte do pacote de serviços oferecido aos usuários.

Figura 1: Computação utilitária

A Figura 1 apresenta o modelo de computação utilitária. Muitos provedores de computação utilitária tinham um *data center* com uma grande quantidade de equipamentos e boa variedade de arquiteturas de servidores, em virtude das diferentes necessidades dos clientes para executar diversos tipos de aplicações. Sendo assim, o cliente requisitava um recurso computacional, informando a configuração e as particularidades para a alocação adequada no sentido de executar sua aplicação. Além do mais, os clientes não tinham tantos recursos para acessar o ambiente do provedor de computação utilitária, pois havia a necessidade de uma conexão direta com o *data center*, ou via internet, mas com a velocidade reduzida e com limitações de serviços.

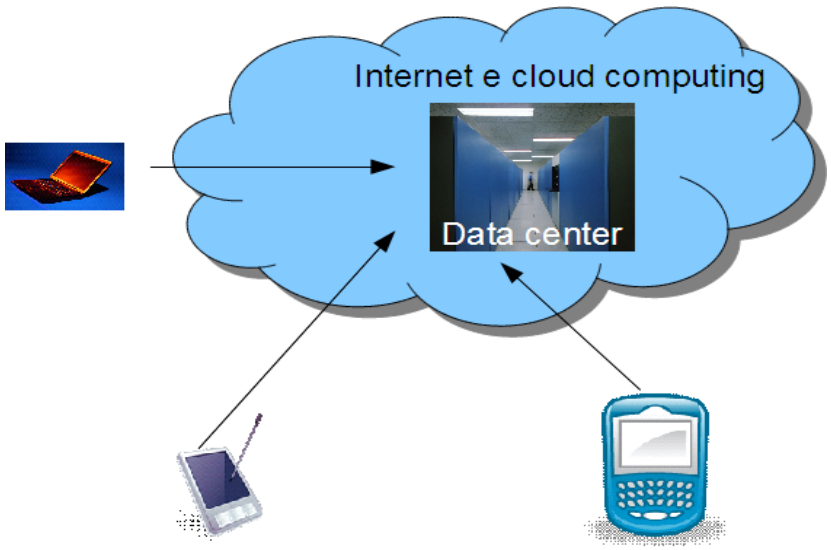

Figura 2: Computação nas nuvens

A Figura 2 mostra como a nuvem está estruturada. Cada ambiente de nuvem encontra-se localizado em *data centers*, assim como ocorreu com a computação utilitária, porém, pelo fato de as aplicações e formas de acessos aos ambientes computacionais serem por meio de diversos métodos, transmite a sensação de que o recurso está conectado diretamente à rede do cliente. Contudo, para chegar a essa infraestrutura, é preciso estar conectado à internet ou ao meio que o provedor da nuvem fornecer. Além do mais, com o desenvolvimento dos dispositivos eletrônicos (PDA, celular, notebook, computador etc.), melhorou a forma de acesso a esse ambiente de nuvem. É preciso apontar, ainda, a existência de outros avanços tecnológicos, como a rede de internet 3G, *wireless*, rede de banda larga etc.

A computação nas nuvens foi conceituada diferentemente da computação utilitária, para que se resolvam problemas de compatibilidade e interoperabilidade de diferentes ambientes computacionais com o mesmo equipamento hospedeiro. Essa possibilidade na computação utilitária não era viável, pois havia a dependência do *hardware* para particionar o equipamento e viabilizar a infraestrutura requisitada pelo cliente. Por outro lado, na computação a dependência do equipamento pode ser solucionado pela tecnologia de virtualização possibilitando que ele fosse executado por meio de uma máquina virtual (MV). Com essa progressão da tecnologia, as infraestruturas foram dividas em três pilares conceituais: infraestrutura (IaaS), plataforma (PaaS) e aplicação (SaaS).

Na IaaS, é possível implementar ferramentas de virtualização, com as quais se pode virtualizar máquinas e executar múltiplas imagens de SO em apenas uma máquina física, sendo necessários recursos disponíveis para alocar e desalocar de acordo com a necessidade da MV. Conforme a MV é criada e configurada, o desempenho do aplicativo pode sofrer quedas, muitas vezes por falta de recursos suficientes. Com uma configuração apropriada, é possível que o desempenho apresente ganhos, por meio de recursos computacionais sem *page fault* ou outros tipos de *overhead*. Portanto, a partir de uma configuração ideal da MV, as aplicações podem ter um comportamento mais eficientes com um processamento sem falta de recursos. Alguns exemplos de IaaS são: Amazon EC2, GoGrid, Flexiscale, OpenNebula, Nimbus.

A PaaS consiste em implementar um sistema operacional e/ou um *framework* de aplicativos que são executados sobre a infraestrutura. O objetivo de criar esses *frameworks* seria fornecer suporte a APIs e a uma plataforma de desenvolvimento com métodos previamente disponibilizados, agilizando, assim, o processo de implementação nesses ambientes, nos quais se pode obter uma plataforma específica para executar uma determinada aplicação. O fato de a aplicação ser desenvolvida em uma plataforma PaaS significaria que tal produto pode ser implementado em qualquer provedor de nuvem que suporte a API do *framework*. Além disso, nesta camada, é possível aplicar recursos para acessar e manipular *storage*, banco de dados, camada de negócios e aplicações web. Alguns exemplos de PaaS: Microsoft Azure, Google AppEngine, Amazon SimpleDB/S3.

Por fim, a SaaS tem como objetivo o processamento efetivo da aplicação. Essas aplicações podem ser desenvolvidas em uma plataforma de desenvolvimento (Java, Python, .NET, Delphi) que tenha suporte no PaaS. Também é preciso ter o suporte IaaS para que seja executado o servidor de aplicação contendo o aplicativo implementado. Somente depois de ter essa definição e implementação é que se pode disponibilizar para os usuários finais o serviço a ser consumido.

Hoje em dia, o termo computação nas nuvens vem sendo explorado em vários aspectos de conceituação, definição, solução, implementação e até em diferentes estratégias corporativas para fazer negócios (*business*) e melhor aproveitar os recursos computacionais da empresa. Esses recursos computacionais, que em épocas de pico encontram-se em funcionamento quase total, têm um custo justificado pelo processamento e pela utilização dos *hardwares*. Contudo, em período de baixa demanda, os programas não ocupam todos os recursos a todo momento. É nessa fase que o *hardware* adquirido – muitas vezes superestimado para suportar as altas demandas – começa a ter seu retorno de investimento (ROI) questionado. A computação nas nuvens tem como propósito buscar, mesmo na baixa demanda, uma melhor utilização dos equipamentos e recursos de TI. Conforme mostra a Figura 3, em geral, a computação nas nuvens é dividida em quatro camadas: *hardware*, infraestrutura, plataforma e aplicação.

- **Camada de *hardware*:** é responsável por gerenciar os recursos físicos da nuvem, podendo ser o servidor físico, roteadores, *switches*, *storages* e qualquer equipamento que faça parte do contexto de computação nas nuvens. Normalmente, os *data centers* possuem diversos equipamentos físicos para serem gerenciados. E os respectivos mecanismos de gerenciamento (*software*), podendo ter balanceamento de carga, tolerância a falhas, configuração remota, gerenciamento de tráfego de dados, entre outros.
- **Camada de infraestrutura:** pode ser chamada também de camada de virtualização, pois essa arquitetura tem o propósito de particionar o recurso físico presente na camada de *hardware*, para implementar uma MV e executar tarefas ou programas com esse recurso físico particionado. Para a virtualização, há diversos gerenciadores de MV (monitores de máquinas virtuais), como, por exemplo: Xen (FOSTER, 1999), KVM (IRFAN, 2008)], VMWare (VMWA, 1999), entre outros.
- **Camada de plataforma:** normalmente, essa camada é implementada sobre a camada de infraestrutura e consiste em um sistema operacional, ou aplicativo, utilizando algum *framework*. Sendo assim, o ambiente no qual os programas serão executados necessitará de bibliotecas, API, servidores de aplicações que requererem um sistema operacional e infraestrutura específicos. Porém, essa arquitetura deve ser explícita para que, na criação da MV, seja designado o equipamento com a arquitetura correta para a sua execução. Portanto, é de suma importância para o provedor das MVs que haja um conhecimento prévio dos aplicativos a serem executados no ambiente virtualizado do seu *data center*.
- **Camada de aplicação**: essa camada consiste na aplicação propriamente dita, em que um *software* ou *web site* é implementado sobre uma plataforma.

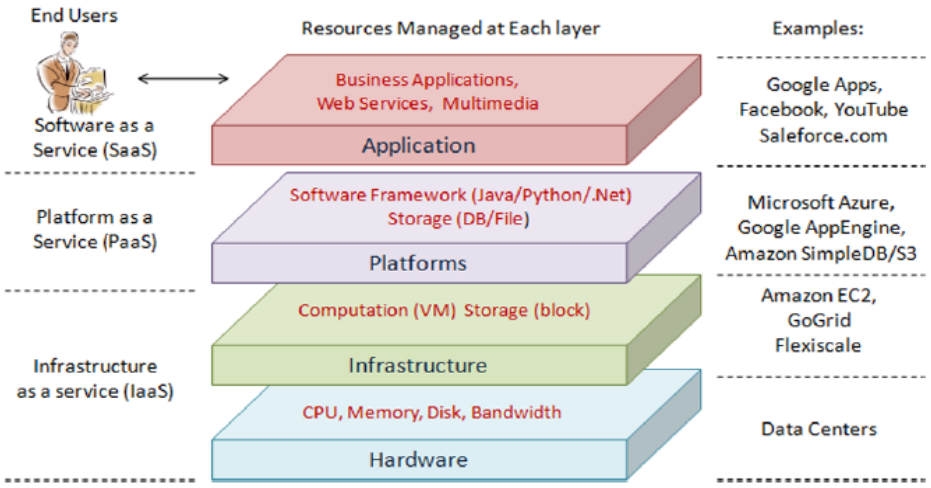

Figura 3: Arquitetura de computação nas nuvens (ZHANG, 2010)

Além da arquitetura da computação nas nuvens, faz-se necessário discutir as diferentes implementações de cada uma das camadas: o IaaS para solucionar a implementação da camada de *hardware* e infraestrutura; o PaaS é o recurso a ser utilizado na camada de plataforma; e, por fim, o SaaS executa as aplicações disponíveis na camada de aplicação.

2.1.1 Tipos de nuvens

Basicamente, computação nas nuvens pode ser classificada em quatro tipos de implementação: pública, privada, híbrida e nuvem privada virtual. Cada um desses tipos possui características a serem identificadas.

2.1.1.1 Nuvem pública

A nuvem pública tem como característica a utilização de recursos computacionais por meio de provedores de serviços, e é disponibilizado uma infraestrutura para o processamento de aplicativos suportados pelo provedor da nuvem pública. A vantagem de implementar uma nuvem pública consiste na prontidão dos recursos computacionais e ambientes previamente configurados e preparados para criar, modificar ou remover

ambientes virtualizados a qualquer momento. Sendo assim, para uma empresa que acaba de nascer, não é necessário adquirir equipamentos e servidores para iniciar sua operação. Tendo um provedor de serviço que possibilite um SaaS ou IaaS (Opex), pode-se obter um ambiente pronto para executar após alguns minutos. Por outro lado, se a empresa é recente e precisa comprar um servidor, faz-se necessário investir na aquisição (Capex) de um equipamento robusto e com capacidade para suportar a demanda dos usuários. Além disso, necessita-se da aquisição de licença de programas para a operacionalização do produto no ambiente de produção. Muitas vezes, essa licença pode ser utilizada por um usuário desse sistema ou por outra modalidade que faça o produto ficar caro e o investimento inicial, maior. Para finalizar os aspectos negativos do Capex, há um prazo de entrega dos equipamentos, *softwares* e da infraestrutura que adiará o funcionamento completo da nova empresa.

2.1.1.2 Nuvem privada

A nuvem privada é um ambiente de computação nas nuvens totalmente local, disposto na infraestrutura da corporação e que utiliza os recursos computacionais do *data center* para o processamento dos aplicativos, particionando o *hardware* e criando máquinas virtuais para necessidades internas da empresa. Com isso, a possibilidade de melhor aproveitar os equipamentos disponíveis na infraestrutura permite que esse equipamento virtualizado seja utilizado de diversas formas. Com essa estratégia, a necessidade de adquirir novos equipamentos não é uma prática tão recorrente, uma vez que novos sistemas podem ser implementados em MVs para evitar todo o processo de aquisição de equipamento, adequando a arquitetura, o sistema operacional, a licença etc, além do processo de instalação do sistema operacional, servidores de aplicações, APIs, entre outras ferramentas necessárias para o bom funcionamento do aplicativo.

2.1.1.3 Nuvem híbrida

A nuvem híbrida é uma combinação de suporte para nuvem pública e privada. A solução da nuvem híbrida tem características interessantes para integrar MVs entre os provedores de serviços (ISP) e a infraestrutura interna da empresa, migrando uma MV

local para um provedor e vice-versa. Essa flexibilidade cria políticas de alocação de recursos na infraestrutura interna e na nuvem pública porque, ao utilizar o ambiente fornecido pelo provedor de nuvem pública, a empresa terá a cobrança pelo tempo de uso. Na indisponibilidade de recursos internos, a política da empresa pode optar pela solução de alocar a MV na infraestrutura da nuvem pública.

2.1.2 Desafios da computação nas nuvens

De fato, a computação nas nuvens precisa melhorar em diversos aspectos em termos de definição e conceituação padronizada. É importante que esse novo paradigma da computação seja explorado e diversificado em outras direções, além das dificuldades que o conceito permite tratar, mas que na implementação ainda são teoria. Entretanto, na atual situação não há um consenso a respeito das melhorias que precisam ser feitas e trabalhadas, mas um dos artigos estudados neste trabalho (ARMBRUST, 2009) apresenta uma lista dos dez obstáculos e oportunidades relativos às melhorias na computação nas nuvens.

	Obstacle	Opportunity
1	Availability of Service	Use Multiple Cloud Providers to provide Business Continuity; Use Elasticity to Defend Against DDOS attacks
2	Data Lock-In	Standardize APIs; Make compatible software available to enable Surge Computing
3	Data Confidentiality and Auditability	Deploy Encryption, VLANs, and Firewalls; Accommodate National Laws via Geographical Data Storage
4	Data Transfer Bottlenecks	FedExing Disks; Data Backup/Archival; Lower WAN Router Costs; Higher Bandwidth LAN Switches
5	Performance Unpredictability	Improved Virtual Machine Support; Flash Memory; Gang Scheduling VMs for HPC apps
6	Scalable Storage	Invent Scalable Store
7	Bugs in Large-Scale Distributed Systems	Invent Debugger that relies on Distributed VMs
8	Scaling Quickly	Invent Auto-Scaler that relies on Machine Learning; Snapshots to encourage Cloud Computing Conservationism
9	Reputation Fate Sharing	Offer reputation-guarding services like those for email
10	Software Licensing	Pay-for-use licenses; Bulk use sales

Figura 4: Os 10 maiores obstáculos e oportunidades de melhorias na computação nas nuvens (ARMBRUST, 2009)

Segundo a Figura 4, depreende-se os obstáculos e desafios para o crescimento da computação nas nuvens. Uma vez adotados esses itens, os dois últimos tornam-se

obstáculos de políticas e negócios para a adoção da computação nas nuvens.

2.1.2.1 Disponibilidade de um serviço

Em uma aplicação crítica a disponibilidade do serviço se torna importante para o funcionamento dos processos e negócios da corporação. Em qualquer um desses sistemas, é necessário fazer uma avaliação para identificação de “pontos únicos de falha”, evitando, assim, a indisponibilidade do serviço requisitado em caso de falha do equipamento. Portanto, em situações de serviços implementados via computação nas nuvens, é importante analisar a forma como o ponto único de falha será tratado, pois, se por algum motivo o provedor de serviço ficar indisponível, o aplicativo, por conseguinte, estará inacessível também. Uma possível solução para esse problema seria contratar dois provedores de serviço e fazer a implementação do aplicativo em ambos. Porém, nem sempre é possível utilizar esse método, porque o custo desse ambiente será elevado em virtude da contratação desses dois serviços. Outra situação seria implementar micronuvens privadas, nas quais apenas os aplicativos importantes seriam disponibilizados, retornando à nuvem pública assim que o provedor voltar a operar.

2.1.2.2 Dados em *lock-in*

Atualmente, os provedores de computação nas nuvens permanecem com APIs proprietárias, sem a possibilidade de interoperabilidade entre as infraestruturas. Até o momento, os provedores não têm tido um diálogo no sentido de padronizar os ambientes de aplicativo ou MVs. Com essa dificuldade, a integração entre diferentes provedores torna-se complexa. A solução para esse obstáculo seria padronizar as APIs para que desenvolvedores de aplicações PaaS possam integrar diferentes aplicativos usando estruturas diversas de nuvem pública.

2.1.2.3 Confidencialidade e auditabilidade

Esse desafio talvez seja o mais complexo e custoso para ser tratado em uma nuvem

pública. Principalmente pela questão da confidencialidade das informações publicadas na infraestrutura do provedor de serviço, que podem vazar e causar prejuízo para a corporação. Portanto, as empresas enxergam a nuvem com cautela ao transferirem os aplicativos para um provedor de serviço. Com relação à auditabilidade, este é um dos obstáculos que precisam ser mais trabalhados, pois não há consenso sobre como seriam feitas auditorias quanto à infraestrutura, acessibilidade, segurança, entre outros tópicos importantes no ambiente computacional do aplicativo.

2.1.2.4 Gargalos na transferência de arquivos

As aplicações corporativas tendem a ter grandes massas de dados que podem, chegar a níveis bastante elevados. Para que o aplicativo seja transferido para a infraestrutura do provedor de nuvem pública, pode haver um gargalo de envio das informações (dados) necessárias para o funcionamento deste aplicativo. Estes dados podem levar algumas horas para serem carregados na infraestrutura do provedor. Esse é um obstáculo e as corporações precisam lidar com esse limitação técnica. Existem algumas propostas para resolver esse problema (ARMBRUST, 2009), mas elas ainda não saíram do plano teórico.

2.1.2.5 Imprevisibilidade no desempenho

Pelo fato de as MVs compartilharem o mesmo recurso computacional (CPU, memória, rede), pode ocorrer *overhead* na utilização de um desses recursos, e, consequentemente, atraso na entrega da resposta do programa. Para obter uma estimativa de desempenho, é necessário ter conhecimento de todos os aplicativos em execução nas MVs, e o *workload* para, enfim, melhorar a previsão de desempenho de uma aplicação.

2.1.2.6 Armazenamento escalável

Uma das propriedades de computação nas nuvens é a sensação de recursos infinitos disponíveis para serem alocados e desalocados. Porém, no caso de armazenamento

(*storage*), este é um assunto que precisa ser explorado pelos provedores da nuvem para que nenhuma das requisições falhe ou faltem recursos para o armazenamento que o cliente da nuvem estiver solicitando. Caso uma demanda por mais espaço não seja atendida, provavelmente a empresa cliente migrará o aplicativo para outro provedor com estabilidade. No caso de demanda por aumento de espaço em disco – o que acontece com grande frequência –, o provedor precisará de mecanismos para escalar o *storage*, o que muitas vezes não é uma tarefa trivial. O grande desafio é definir e implementar políticas de fornecimento de recursos de armazenamento de dados para os clientes da nuvem. Essa dificuldade permanece como um tópico em aberto em todos os provedores de serviço, porque atualmente eles alocam e desalocam os espaços de acordo com a demanda solicitada pelos clientes.

2.1.2.7 *Bugs* em larga escala em sistemas distribuídos

Desenvolver sistemas distribuídos já é um desafio por natureza, pois o comportamento de aplicações varia conforme a resposta de cada nó do *cluster*. Além de desenvolver mecanismos de tolerância a falhas, exclusão mútua, integridade de dados, entre outros, faz-se necessária a abstração de sintomas de comportamento do aplicativo em ambientes de larga escala. Normalmente, o desenvolvimento desse aplicativo distribuído ocorre em ambientes menores em termos de *hardware* (apenas para testes, e análises de desempenho por exemplo), simulando o ambiente de produção. No entanto, a implementação em ambiente de produção pode resultar em situações que não foram identificadas ou previstas na fase de desenvolvimento, e isso dificulta a finalização dos processos distribuídos nas MVs implementadas em um provedor de computação nas nuvens.

2.1.2.8 Escalonando rapidamente

Pelo fato de a computação nas nuvens se dar em um modelo de Opex, as necessidades de mais recursos computacionais relacionam-se proporcionalmente à demanda do aplicativo e às requisições dos usuários do programa. Da mesma forma, a necessidade de desalocação de recusos computacionais torna-se um custo operacional

para o provedor, pois o sistema inativo (*idle*) representa um recurso computacional disponível para processar, porém não há processamento. O custo operacional é sempre um fator negativo para o provedor de serviço, porque é preciso utilizar os recursos computacionais com planejamento, deixando o processamento para os aplicativos ativos e desalocando recursos quando não há mais utilização. Ocorre o mesmo fenômeno com a alocação de recursos, na qual, é possível criar políticas de alocação e desalocação, permitindo aumentar a capacidade de processamento de uma MV. Assim que se finaliza a execução, volta-se para a situação inicial, liberando o recurso para ser alocado em outra MV.

2.1.2.9 Lista de reputação

Um dos artigos estudados (ARMBRUST, 2009) trata esse tema como *Reputation Fate Sharing* (compartilhamento de reputação), nada mais do que uma lista de reputação por cliente. Uma vez que a nuvem tem o propósito de alocar uma MV na infraestrutura do provedor, pode-se utilizar essa MV para assuntos ilegais, invasões, *spamming*, entre outros comportamentos inadequados. Para evitar esse tipo de problema, o provedor precisa de mecanismos para evitar que aplicativos dessa natureza possam afetar seu ambiente como um todo, além de meios de identificação e isolamento da comunicação, para evitar que os IPs desse provedor entrem em *blacklists* (lista de IPs que tem o intuito de bloquear acessos suspeitos de IPs utilizados para invasão ou ato ilícito na internet). Outra forma de resguardar a integridade do provedor consiste em criar documentos de transferência de responsabilidade, nos quais o usuário responderá por qualquer processo por práticas ilícitas, que não condizem com a política de segurança do provedor de serviço.

2.1.2.10 Licença de software

Com relação às licenças de *softwares*, ainda há o que ser explorado pelas empresas de desenvolvimento de programas, uma vez que há compatibilidade do método *pay-as-you-go* com o modelo de negócios de vendas por quartil, que é o modelo de negócios empresariais norte americano. Portanto, as empresas teriam de readequar o modelo de

vendas para, então, ser possível implementá-lo em ambientes de computação nas nuvens. Por enquanto, as empresas têm utilizado ferramentas *open-source* ou possuem a licença de uso anterior.

2.2 VIRTUALIZAÇÃO

A virtualização é uma tecnologia que foi aplicada em diversos programas computacionais nos quais se objetivava simular o comportamento de uma máquina ou programa por meio de uma máquina real. Apesar dessa máquina ou programa virtual não se encontrar implementada diretamente na máquina real, isso não significava que ela estivesse inacessível. Muitas vezes, para o usuário era transparente a utilização desse ambiente virtual, pois ele aparentava estar disponível mesmo sem existir fisicamente (LAUREANO, 2006). Mas, por intermédio desta virtualização, pode-se criar aplicativos que executam instruções especificas de uma linguagem de programação ou plataforma em diferentes sistemas operacionais (SO) de uma máquina real; por exemplo: Java Virtual Machine (ORLANDO, 2006). Além desta solução, pode-se virtualizar sistemas operacionais, transformando-os em processos do sistema operacional gerenciado por meio de um monitor (VMM) ou *hypervisor*. Esta abstração das máquinas virtuais permite que diferentes SOs possam ser criados sobre uma mesma máquina física, virtualizando os recursos computacionais e dividindo-os entre cada MV.

2.2.1 Tipos de monitores de máquinas virtuais

Os monitores de máquinas virtuais (VMMs) são responsáveis pelo gerenciamento e processamento das MVs, sendo uma camada intermediária entre o sistema operacional da máquina real e a máquina virtual. Uma vez que o monitor influencia no processamento das aplicações, é preciso que exista um meio de classificar tipos diferentes de MMVs. Nos tópicos seguintes, apresentamos três tipos de monitores de máquinas virtuais.

2.2.1.1 MMV - tipo I

Os monitores de máquinas virtuais do tipo I são implementados entre o *hardware* e o sistema operacional, com acesso e controle direto ao *hardware*. Esta arquitetura permite que as máquinas virtuais possam executar aplicativos como se o usuário estivesse em uma máquina real, pois as instruções serão processadas diretamente no *hardware*.

A Figura 5 ilustra a forma de organização das MVs em um computador real. O *hardware* está logo abaixo do monitor, que recebe as solicitações de instruções requeridas das aplicações em execução nas MVs.

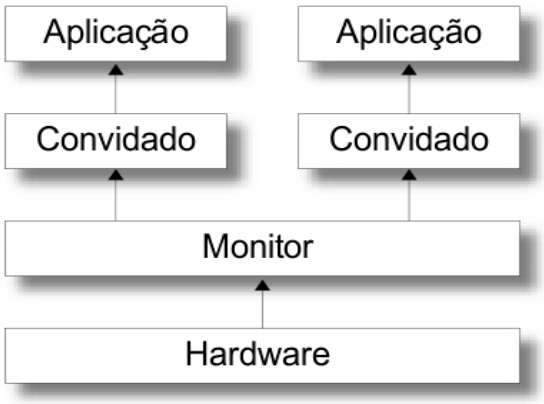

Figura 5: Monitores de máquina virtual do tipo I (LAUREANO, 2006)

2.2.1.2 MMV - tipo II

Os monitores de máquinas virtuais do tipo II são implementados por meio de um processo comum no sistema operacional anfitrião. Esse processo, por sua vez, instancia as requisições de processamento necessárias para a aplicação nas MVs e transfere o processamento para o sistema operacional real. Neste modelo, todos os processamentos e interrupções de *hardware* serão feitos sempre por intermédio desse processo, impossibilitando otimizações. Obrigatoriamente as comunicações e solicitações de processamento ou interrupção de *hardware* serão feitas por meio do processo MMV.

A Figura 6 apresenta o modelo de arquitetura MMV do tipo II, sendo que o sistema operacional anfitrião está na interface entre o *hardware* e o monitor. Portanto, esse modelo possui uma camada a mais, muitas vezes havendo um atraso no processamento por conta desta arquitetura.

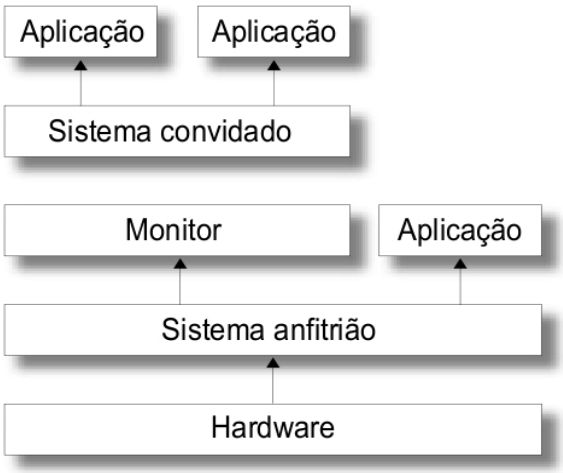

Figura 6: Monitores de máquinas virtuais tipo II (LAUREANO, 2006)

2.2.1.3 MMV - Híbrido

O monitor de MV híbrido é caracterizado por otimizar os monitores dos tipos I e II, alterando o método de implementação e possibilitando meios para eliminar a sobrecarga do monitor.

A otimização do MMV do tipo I foi a modificação do núcleo (*kernel*) do sistema operacional anfitrião e da MV. Uma vez que ambos os sistemas operacionais estejam com o mesmo núcleo, há a eliminação de uma camada de processamento. Sendo assim, o *hardware* pode ser controlado pelo SO da MV, otimizando sua alocação sem que haja interferência do monitor de máquinas virtuais.

A Figura 7 apresenta o modelo de otimização do monitor de MV híbrido, com uma via (1) ligando a máquina virtual diretamente ao *hardware*, sem a interferência do monitor.

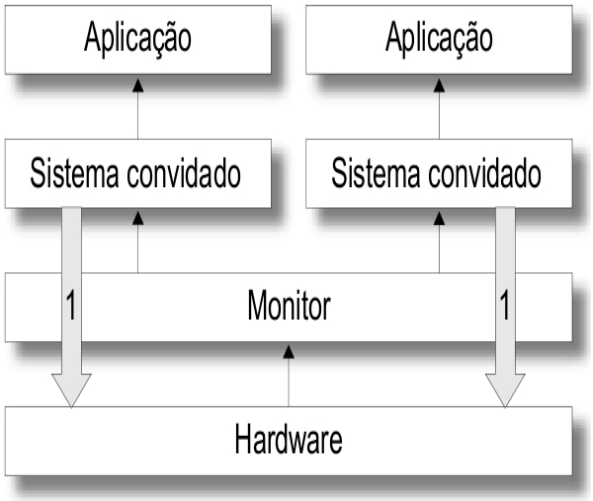

Figura 7: Monitores de máquinas virtuais híbridas tipo I (LAUREANO, 2006)

Já na modificação do monitor de máquinas virtuais do tipo II, é aplicada a mesma abordagem do híbrido tipo I, porém com outras soluções de otimizações e alocação de *hardware*. A Figura 8 apresenta os métodos de otimização para a comunicação das camadas. No primeiro método (1), o sistema convidado (MV) acessa diretamente o sistema operacional da máquina anfitriã, eliminando o *overhead* dos monitores. Outra forma de otimização (2) é o sistema convidado (MV) acessar diretamente o *hardware*, sendo que esta implementação precisa ser feita parcialmente pelo sistema operacional anfitrião e o MMV. Alguns exemplos de dispositivos que podem ser utilizados neste modelo: *drivers* de CDs, placa de vídeo, placa de rede, entre outros. E por fim (3), o método de acesso direto do MMV ao *hardware*. Este modelo permite que o MMV possa ter acesso a dispositivos oferecendo uma interface de baixo nível, além de possibilitar acesso direto ao *hardware* quando o monitor demandar a interrupção.

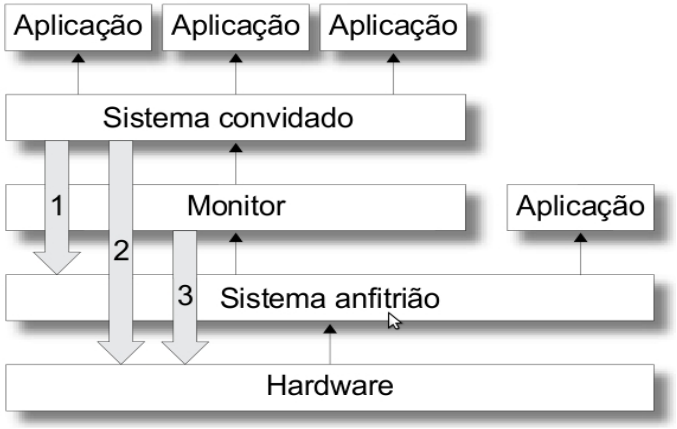

Figura 8: Monitores de máquinas virtuais híbridas tipo II (LAUREANO, 2006)

2.2.2 Monitores de máquinas virtuais

Esta seção apresenta três aplicações de monitores de máquinas virtuais difundidos nos ambientes corporativos e acadêmicos. O objetivo desta apresentação dos MMVs é caracterizar cada aplicação dentro dos conceitos e tipos de MMVs vistos na seção anterior (seção 2.2.1) e introduzir o Xen como parte do experimento aplicado nesta dissertação.

2.2.2.1 Xen

O Xen (FOSTER, 2008) é um MMV altamente aceito pela comunidade acadêmica, pelo fato de estar inserido no contexto de monitores de MV tipo I, ou seja, as MVs têm o mesmo núcleo (*kernel*) da máquina real. Adicionalmente, o Xen é *open source* e conta com muitos adeptos na comunidade mundial; muitos artigos acadêmicos estão disponibilizados para exploração e aprimoramento deste monitor. O Xen tem suporte a dois modelos de virtualização: *fullvirtualization* e *paravirtualization*.

A *fullvirtualization* faz com que a MV funcione exatamente como uma máquina física, com memória, CPU e alocação de HD, sempre por meio do *hypervisor* (BARHAM, 2003). Em outras palavras, a imagem da MV usa uma cópia exata de um sistema operacional, sem qualquer tipo de manipulação ou modificações de recursos,

drivers, *kernel*, API etc. Essa estrutura pode ter trocas de contextos e *traps*, o que aumenta a complexidade e pode diminuir o desempenho por conta do *overhead* dessa tarefa.

Por outro lado, a *paravirtualization* utiliza um *kernel* modificado para fazer comunicação com o *hypervisor* no intuito de interagir diretamente com o *hardware*, não havendo intervenção do *kernel* do sistema operacional da máquina hospedeira. Desse modo, uma imagem em *paravirtualization* tem um desempenho elevado por conta de não ter um intermediário interpretando e enviando mensagens para o sistema operacional real. Além disso, ter um *kernel paravirtualization* modificado faz com que uma imagem paravirtualizada seja independente do sistema operacional da máquina hospedeira.

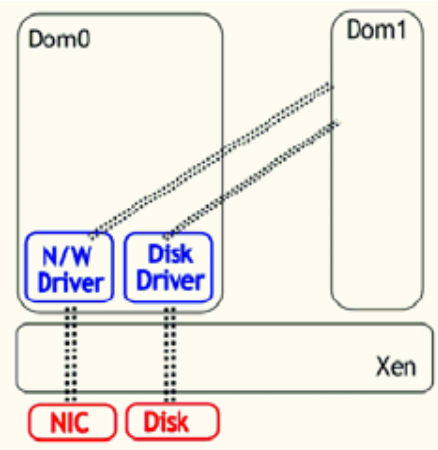

Figura 9: ***Arquitetura atual de I/O no XEN (CHERKASOVA, 2007)***

A Figura 9 mostra o modelo atual de I/O no Xen, em que o domínio 0 (dom0) é o responsável por conter todos os *drivers* dos dispositivos no sistema operacional, executando todas as requisições no sistema operacional hospedeiro. Com esse modelo, a rede e o I/O de disco são gerenciados pelo dom0, sendo assim, as MVs necessitam fazer uma solicitação para que o dom0 efetue a comunicação ou I/O de disco. Neste caso, o desempenho da rede e do I/O de disco será mais eficiente, fazendo com que o dom0 implemente a comunicação sem passar pelas camadas de

rede. Usar MVs na mesma máquina hospedeira faz com que a comunicação entre as MVs tenha um desempenho maior. Se forem executadas em máquinas diferentes, ocorrerá comunicação com a rede física e, portanto, haverá atraso no processamento de uma MV com outra.

2.2.2.2 VMware

O VMware é um monitor de máquinas virtuais implementado por meio de uma camada de software (*VMware Virtual Platform*) (VMWARE, 1999). Essa camada intermediária possibilita que o VMware funcione de duas maneiras, MMV do tipo I (Figura 10, quadro 1), executando diretamente no *hardware*; ou MMV do tipo II (Figura 10, quadro 2), implementado em um sistema operacional, funcionando como um processo tradicional.

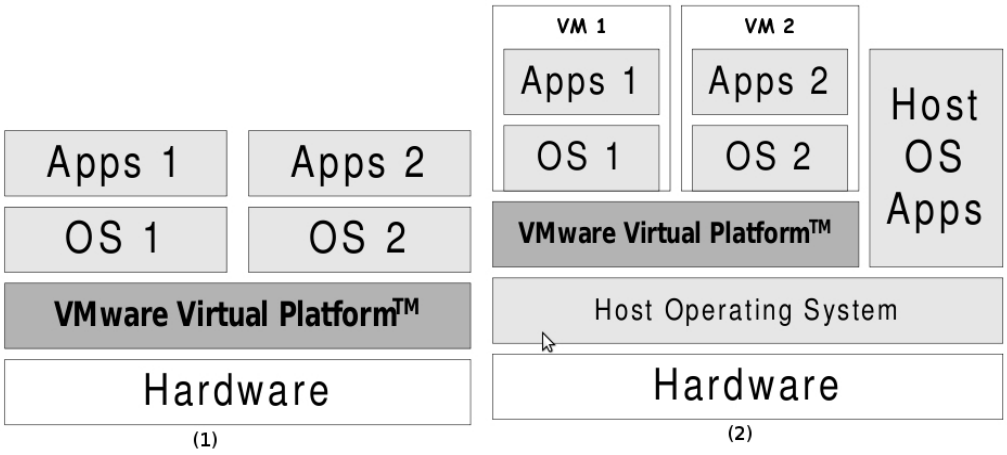

Figura 10: Arquitetura VMware (VMWARE, 1999)

Adicionalmente, o *VMware Virtual Platform* possui três componentes de software que têm por objetivo diminuir o *overhead* de processamento nas MVs:

- *Virtual Platform* (VP) *Application*: nesta arquitetura, a aplicação virtual tem o comportamento de um programa tradicional em um sistema operacional. Contudo, é implementada por meio de uma MV, sendo transparente para o usuário do aplicativo. Para qualquer requisição ou interrupção de

hardware, o MMV deverá fazer o processamento;

- *Virtual Platform Monitor*: o VP *monitor* possui privilégios para executar diretamente no *hardware*, o que elimina o *overhead* da técnica de virtualização, como acontece com o *Virtual Platform Application*. O VP *monitor* foi especificado para executar na arquitetura X86, e esta limitado a essa arquitetura de processadores. Contudo, nas operações de *system resources*, é mantida a responsabilidade do sistema operacional, incluindo gerenciamento de memória física e escalonamento da CPU;
- *Virtual Platform Driver*: o VP *driver* é um recurso que inclui os *devices drivers* no próprio sistema operacional. Esses *drivers* atuam como um *gateway* de comunicação entre as aplicações e o monitor. Com isso, o VP *driver* aparenta ser um processo tradicional para o sistema operacional, mas efetivamente faz a comunicação direta com o dispositivo.

2.2.2.3 Kernel virtual machine (KVM)

O KVM é um MMV nativo do sistema operacional Linux que suporta as arquiteturas x86. Conforme a Figura 11, o KVM é apresentado como MMV do tipo I, sendo que é criado um *device driver* (`/dev/kvm`) no qual, para cada MV, é gerado um processo no sistema operacional hospedeiro e alocados os recursos de memória e disco. Contudo, o escalonamento de CPU permanece gerenciado pelo SO da máquina real. As operações do `/dev/kvm` incluem: criação de uma nova máquina virtual, alocação de memória para a MV, leitura e gravação virtual dos registradores de CPU, interrupção da CPU virtual e execução do processamento na CPU virtual.

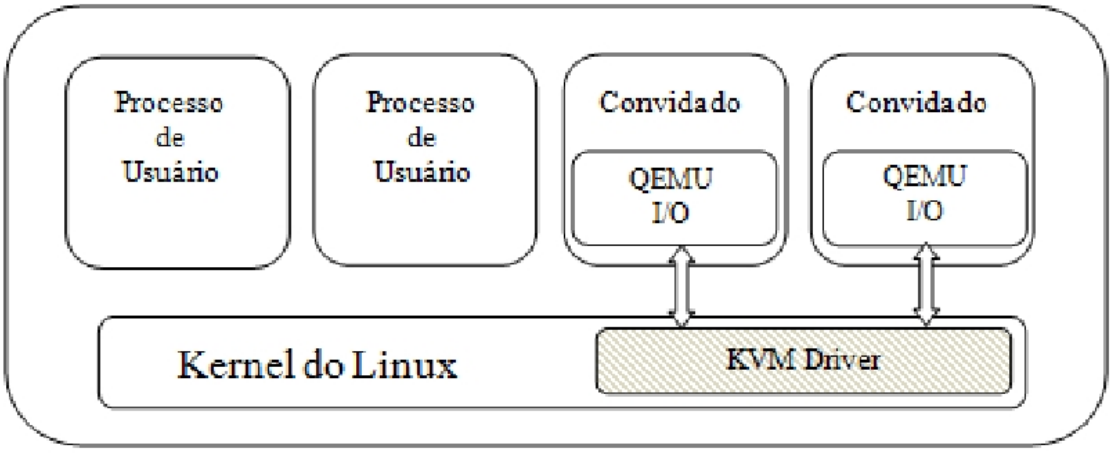

Figura 11: Arquitetura KVM (BARUCHI, 2010)

2.2.3 Computação nas nuvens privada

No modelo de computação nas nuvens privada, pode-se usar a ferramenta OpenNebula (SOTOMAYOR, 2008), desenvolvida para funcionar sobre a arquitetura IaaS, para gerenciar as MVs por meio de uma única aplicação. por intermédio do gerenciador do OpenNebula pode-se criar, migrar, remover, monitorar e controlar MVs. Contudo, o OpenNebula necessita de uma ferramenta de monitoramento de monitores de máquinas virtuais (VMM) que possa implementar um sistema operacional em MVs, sendo que suporta três monitores de MVs: Xen, VMWare e KVM.

O OpenNebula pode ser implementado nas três arquiteturas de computação nas nuvens: pública, privada e híbrida. Com isso, possibilita mover ou implementar uma MV facilmente em diferentes infraestruturas, podendo ou não utilizar recursos externos à corporação. Essa flexibilidade possibilita que os gestores da corporação optem pelos serviços dos provedores e, posteriormente, sejam cobrados pelos minutos utilizados pela MV ou aplicação que utilizou uma nuvem pública. Com alguns comandos e uma configuração apropriada para alcançar uma nuvem pública, pode-se migrar um ambiente inteiro de uma nuvem privada.

A Tabela 1 apresenta os recursos disponíveis do OpenNebula:

Tabela 1: Recursos do OpenNebula

Recurso	Descrição
Escalonador	Responsável por distribuir a carga de trabalho entre os nós hospedeiros. O OpenNebula implementa um escalonador simples (mm_sched), atribuindo o nó do OpenNebula que tiver mais recursos computacionais disponíveis (memória, CPU, disco rígido) para executar a MV. Caso não haja disponibilidade, o mm_sched não implementa a MV enquanto o hospedeiro não disponibilizar recursos mínimos para executar a configuração requisitada. É possível utilizar uma ferramenta que substitui o mm_sched, o Haizea (HAIZEA, 2010), a qual também é uma ferramenta *open source* que gerencia a alocação e desalocação de *hardware*.
Gerenciador de rede	O gerenciador de rede das redes virtuais tem o propósito de fazer a intercomunicação entre as MVs. Possui um algoritmo otimizador, o que evita que a comunicação entre as MVs e a máquina hospedeira não seja via TCP/IP. Sendo assim, esse algoritmo utiliza a própria memória compartilhada para fazer a comunicação entre as MVs e máquinas hospedeiras, e entre elas também.
Contextualizador e gerenciador de serviço	Suportam multicamadas de serviços das MVs que estejam intercomunicáveis e um arquivo de contexto que passa as configurações da MV, podendo, assim, padronizar as configurações e promover uma autoconfiguração no *boot* da MV.
Segurança	A segurança é feita por meio do administrador da infraestrutura do nó do OpenNebula. Na máquina hospedeira, podem-se criar contas automáticas para usar em um arquivo de contexto.
Tolerância a falhas	Todas as informações e configurações do OpenNebula estão armazenadas em um banco de dados persistente que

	armazena dados da máquina hospedeira e das MVs.
Escalabilidade	O OpenNebula tem sido avaliado por centenas de servidores e MVs. A possibilidade de criar, implementar, controlar e desligar por meio do OpenNebula faz com que o gerenciamento das MVs seja uma tarefa fácil.

2.3 INSTRUMENTAÇÃO DE APLICAÇÕES

Nessa sessão, é apresentado o conceito de instrumentação de aplicações, um processo fundamental para uma avaliação do desempenho de uma aplicação. Por intermédio de *benchmarks*, são extraídas amostras de determinadas métricas escolhidas para a avaliação de desempenho. As métricas são os critérios das medidas que serão quantificadas e utilizadas para a comparação ou avaliação de desempenho das aplicações; é importante haver um estudo direcionado na aplicabilidade dessa métrica nos cenários a serem avaliados, pois uma métrica indevida ou mal formulada pode afetar toda a análise ou levar a conclusões errôneas. Essas métricas precisam estar de acordo com o propósito definido para análises de carga de trabalho (*workload*), tempo de execução e vazão de dados, entre outras métricas. Cada *benchmark* é modelado e definido para fornecer informações que serão utilizadas na comparação entre os sistemas avaliados. O objetivo da comparação dessas métricas é classificar os aplicativos mais ou menos eficientes dentre um conjunto já instrumentado. Um fator importante a ser considerado é que o ambiente e as configurações a que serão submetidas as aplicações obrigatoriamente precisam estar padronizados para que elas sejam processados sobre as mesmas condições. Para cada aplicativo, é preciso avaliar se o método de aplicabilidade destes *benchmarks* apresenta informações quantitativas e qualitativas capazes de mensurar se um determinado aplicativo teve ou não o desempenho esperado. As métricas quantitativas são os números que representam o consumo de um determinado recurso. É possível utilizar essas amostras numéricas e totalizar as medidas coletadas. Os dados qualitativos são cenários usados para avaliar a qualidade ou a apuração de determinadas situações. Para um estudo completo e detalhado, pode-se utilizar mais de uma métrica para justificar uma determinada medida

que, após analise individual, não obteve base suficiente para levar a alguma conclusão. Porém, é preciso haver um cuidado a mais para que não haja conclusões indevidas, pois as métricas precisam ser complementares e terem um significado coerente. Informações desconexas e não conclusivas devem ser avaliadas separadamente para não afetar o resultado da análise.

Após as métricas serem definidas, é preciso configurar os parâmetros dos limites para que as medidas possam ser caracterizadas e classificadas. Posteriormente, a totalização dessas medidas pode indicar se uma dada informação oriunda dessas métricas foi satisfatória ou não. Estes limites devem ser criteriosos para alcançar uma base analítica justificável.

Por fim, é preciso fazer a coleta das informações das métricas no ambiente real ou simulado. Os aplicativos que possuem infraestrutura e meios para executar em ambientes reais podem ser instalados e executados diretamente em um computador ou MV. Caso a aplicação tenha características específicas e houver inviabilidade para execução em uma máquina real ou sistema operacional suportados, buscam-se alternativas em ambientes de simulação. Para essa situação, é criado um ambiente compatível com os requisitos da aplicação, reproduzindo os resultados com base nesta simulação. Muitas vezes, os resultados podem comportar diferenças significativas se aplicados no ambiente real, por haver variáveis que não foram consideradas na definição do experimento simulado. Como resultado, é possível utilizar dados quantitativos coletados e gerar gráficos, tabelas e relatórios.

A Figura 12 (JAIN, 1991), é apresentada uma proposta com as três possibilidades de resposta de uma requisição de serviço. Sendo que o serviço pode ser processado corretamente (esperado), incorretamente ou recusado (inesperado). A requisição que finalizar o processamento corretamente é categorizada como métricas de velocidades (*response time*), vazão (*throughput*) e alocação de um recurso (*utilization*).

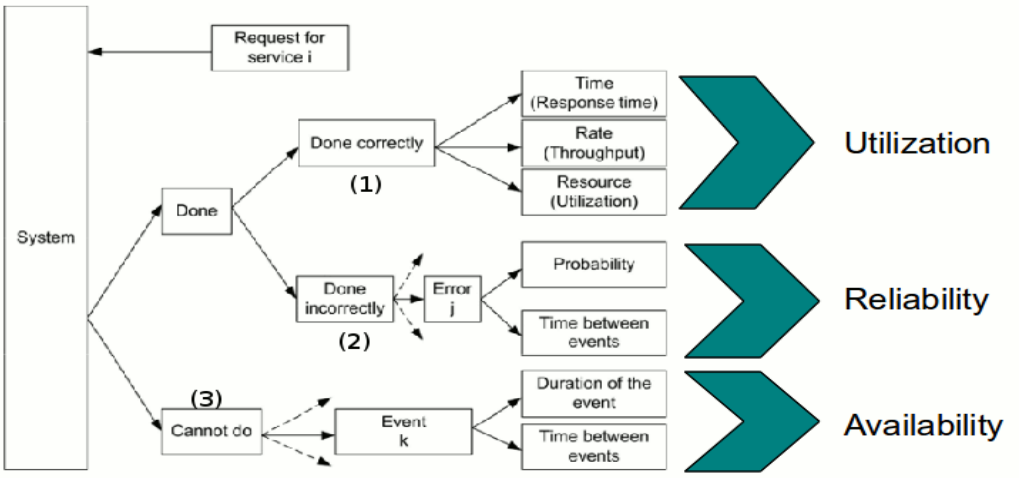

Figura 12: As três possibilidades de uma requisição de serviço (JAIN, 1991)

A Figura 12 apresenta as métricas de avaliação de uma requisição de serviço. Segundo Jain (Jain, 1991), um serviço tem a possibilidade de finalizar com sucesso (comportamento esperado, indicação 1) ou completar a requisição, porém de maneira incorreta (inesperado, indicação 2), ou mesmo ser impossibilitado de processar (inesperado, indicação 3). A seguir, será apresentada uma descrição de cada métrica (1) (2) (3).

Na primeira situação (1), a requisição de processamento de um serviço é recebida e processada corretamente. A seguir, as três métricas de utilização do recurso:

- Tempo de resposta (*response time*): é o intervalo entre a requisição do usuário e a resposta da aplicação. Porém, é possível avaliar outros intervalos de tempo para apurar outras medidas de tempo para situações diferentes, além de ser possível avaliar o tempo desde a requisição do usuário até o inicio da execução do programa (tempo de reação);
- Utilização: cada aplicação é processada em recursos computacionais assim que o usuário requisita o serviço. Os recursos que forem alocados por um período de tempo serão utilizados para calcular a métrica de utilização. No tempo em que o recurso ficou alocado e no qual não houve processamento, é calculado o percentual de utilização. Por exemplo: um

sistema de *enterprise resource planning* (ERP) está disponível vinte e quatro horas por dia, porém foi utilizado 12 horas em média por dia. Portanto, houve 50% de utilização do sistema;

- Vazão de dados: a vazão de dados (*throughput*) é uma métrica que define a taxa de transferência de uma informação (por unidade de tempo). Para determinados aplicativos, indica-se o *throughput* como métrica de análise de um sistema, pois ele indica a capacidade de envio das informações pelo meio de comunicação. Por exemplo: para CPUs, o *throughput* é medido por *millions of instructions per second* (MIPS), ou *million of floating-point operations per second* (MFLOPS). Para redes de computadores, é calculado pelo *packets per second* (pps) ou *bits per second* (bps). Para aplicações transacionais, é calculado por *transactions per second* (TPS).

Agora, a segunda possibilidade (2), na qual a requisição do usuário é recebida com sucesso, porém finaliza com algum erro. Uma medida deve avaliar a probabilidade de ocorrer um erro, ou o tempo em que ocorreu a falha. Para se obter a métrica de probabilidade de ocorrência do erro será necessário efetuar uma avaliação do percentual em que a falha pode ocorrer; ela servirá de parâmetro para indicar se a medida deve ser simplesmente ignorada ou se é preciso avaliar detalhadamente o motivo do erro.

$$Ps = Pa + Pb + ... + Pn$$ (AGARWAL, 2008)

Em que:

- Ps = Probabilidade do sistema falhar; Pa = Probabilidade do componente A falhar; Pb = Probabilidade do componente B falhar; Pn = Probabilidade do componente N falhar.

Outra forma de métrica (2) é calcular o tempo *mean time to Fail (*MTTF), um valor médio calculado com base no tempo de vida de um *hardware*. Conforme o equipamento

vai envelhecendo, o valor de MTTF diminui. Um MTTF de 50 horas significa que a cada 50 horas é esperado uma falha no componente. Outra forma de avaliar é por meio da métrica *mean time to repair* (MTTR), segundo a qual, na ocorrência da falha, é preciso haver um tempo para a resolução do problema. Portanto, o tempo em que houve a falha e a recuperação é a medida do *mean time between error* (MTBE).

$$MTBE = MTTF + MTTR$$ (AGARWAL, 2008)

E, por fim (3), a métrica de disponibilidade (*Availability*). Ela indica a probabilidade de um sistema estar disponível para receber requisições e transações dos usuários. Na situação de indisponibilidade do aplicativo, são analisadas duas medidas: *duration of the event* e *time between events.* A primeira métrica é representada pelo tempo que o evento levou para mostrar a falha (MTTF) e recuperar a situação normal (MTTR). Já a segunda é a soma do MTTF e MTTR do evento, chamada de *mean time between fail* (MTBF).

$$MTBF = MTTF + MTTR$$ (AGARWAL, 2008)

Uma vez calculado o MTFB, pode-se aplicar a fórmula abaixo para descobrir a disponibilidade do ambiente em que foram submetidas as métricas de disponibilidade. Esse percentual refere-se ao tempo em que o sistema ou ambiente esteve em operação por um determinado período. Essa métrica é importante para determinar se um ambiente esta estável ou instável, além de apresentar dados para os clientes deste sistema.

$$Avaliability = \frac{MTTF}{(MTBF)} * 100$$ (AGARWAL, 2008)

2.4 Lógica Fuzzy

A lógica Fuzzy foi estruturada pelo professor Lotfi Zadeh da Universidade de Berkley, Califórnia (LEE, 1990). Zadeh conceituou um método para representar o tratamento de classificação de grupos, porém as regras dos agrupamentos eram incertas. Por exemplo: 40 ºC < temperatura < 60 ºC, ou seja, valores que estejam entre 40 ºC e 60 ºC, ou temperatura do que esteja no agrupamento gelado ou temperatura do que esteja no quente ou no superquente. Os valores de limites são dependentes de uma variável que limita as margens; com este conceito, tornou-se possível definir faixas de valores. Além do mais, utilizando a teoria de conjuntos com a teoria da lógica Fuzzy, é possível definir as regras e operações (união, intersecção, complemento, cartesiano) para cada taxonomia definida por esta lógica. Para uma implementação da lógica Fuzzy, o primeiro passo é decidir o que será controlado e a forma como será administrado. Isso definido, o segundo passo é a determinação das faixas de valores para cada grupo, criando as áreas mínimas e máximas para cada região. Essa é a parte mais difícil da lógica Fuzzy, pois tal delimitação pode variar de acordo com determinadas situações que, em certo momento, tinham um valor baixo, mas, em outro, passaram a ter um valor médio. Assim como outro valor pode ser médio, mas tornar-se baixo, pois valores maiores entraram no método e fizeram-no passar para a faixa de baixo. Esse tipo de situação pode ocorrer e tem comportamento próximo ao do mundo natural. Outra forma de classificar os valores nas faixas é por meio do método da lógica Crisp, que possibilita incluir métodos matemáticos, utilizando conjuntos matemáticos para definir as regiões de cada taxonomia.

2.5 CONCLUSÃO DO CAPÍTULO

Neste Capítulo 2 foi apresentado os conceitos necessários para apoiar a metodologia proposta nesta dissertação. A seguir, será apresentada a metodologia de caracterização de aplicações.

3. METODOLOGIA DE CARACTERIZAÇÃO DE APLICAÇÕES

Uma metodologia tem o objetivo de conceituar e estudar diferentes métodos para o processamento um determinado processo de uma aplicação. Para ser viável a aplicação de uma metodologia, é preciso ter regras e processos. A metodologia tem um papel importante na definição e conceituação do experimento, ou avaliação, pois permite criar as regras de cada situação que identifica os limites, as variáveis comparativas, e até concluir se o método é adequados ou não. A seguir, serão apresentados os componentes da metodologia proposta para caracterizar aplicações em ambiente de computação em nuvens.

3.1 APLICAÇÕES EXPERIMENTADAS

Nesta seção, são apresentadas as aplicações selecionadas para o processo de instrumentação da metodologia de coleta de dados quantitativos, assim como a caracterização de aplicações e a justificativa da adoção deste método para o experimento. Por fim, é apresentado um estudo inicial, em que foram analisadas aplicações paralelas, distribuídas e transacionais em um ambiente de computação nas nuvens. Neste estudo preliminar, foram coletados dados que as justificam a adoção de determinadas configurações de MV para cada característica de aplicação.

3.1.1 Pacote sysstat: comandos de instrumentação

O pacote `sysstat` é um produto que contém aplicativos de coletas de dados do *kernel*, porém não vem originalmente na maioria das distribuições do Linux. Todavia, boa parte das distribuições tem suporte para o funcionamento completo desse pacote. O sysstat possui utilitários de monitoração de desempenho de sistema e utilização de recurso, coletando informações diretamente do `/proc`, diretório virtual que mantém as informações referentes ao *kernel* do Linux. Essas informações são alimentadas pelo próprio *kernel*, apresentando os valores atuais do sistema operacional em seu diretório virtual. A primeira leitura de qualquer aplicativo do “sysstat” apresentará uma média das

informações desde o último *boot*. A partir da segunda leitura, as métricas são reiniciadas para que as próximas informações correspondam a dados mais rescentes. Importante ressaltar que, se não forem isoladas as interferências de outros aplicativos ou dispositivos que não têm relação com o que está sendo avaliado no experimento, o resultado da coleta das métricas na instrumentação poderá ser afetado. O ideal é que tais dados sejam isolados em *file systems* (FS) diferentes ou até fisicamente em discos (HD) diferentes. Sendo assim, a execução do programa e a instrumentação ocorrerão isoladamente de todos os outros processos, ficando apenas a concorrência com o próprio sistema operacional.

3.1.2 Aplicativo `iostat`

O comando `iostat` apresenta os dados estatísticos de *central processing unit* (CPU) e *input-output* (I/O) de um dispositivo ou partição utilizado por uma aplicação, coletando seus dados estatísticos. As saídas geradas pelo `iostat` são detalhadas a seguir.

3.1.2.1 Relatório de utilização de CPU

Esse relatório é apresentado logo no cabeçalho da saída do comando, que captura as situações e o comportamento das CPUs. No caso de multiprocessadores, os valores são apresentados como médias aritméticas entre os consumos dos processadores. A Tabela 2 descreve cada métrica de CPU.

Tabela 2: Saída do comando `iostat`

Métrica	Descrição
`%user`	Mostra o percentual de utilização de CPU que ocorreu durante a execução da aplicação no nível de usuário.
`%nice`	Mostra o percentual de utilização de CPU que ocorreu durante a execução da aplicação no nível de usuário com prioridade `nice`.
`%system`	Mostra o percentual de utilização de CPU que ocorreu durante a execução da aplicação no nível de sistema (*kernel*).
`%iowait`	Mostra a porcentagem do tempo em que as CPUs estiveram no

	estado *idle*, enquanto o sistema atendia a uma solicitação de I/O intenso de disco.
`%steal`	Mostra a porcentagem do tempo gasto na espera involuntária pelas CPUs virtuais, enquanto o *hypervisor* estava servindo outro processador virtual.
`%idle`	Mostra a porcentagem do tempo em que as CPUs estiveram no estado *idle* e o sistema não teve uma requisição de I/O intenso de disco.

O exemplo da Tabela 3 apresenta uma leitura de aplicação por intermédio do comando `iostat` e a CPU esteve disponível em 80% (*idle*), e apenas 12% de processamento no nível do usuário, desde a última leitura do `iostat`.

Tabela 3: exemplo de `iostat` - utilização de CPU

```
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          12.81    0.00    4.34    2.58    0.03   80.25
```

3.1.2.2 Relatório de utilização de dispositivos

O segundo bloco de informações apresentadas no *output* do `iostat` refere-se aos dados coletados a partir de dispositivos. Esses dados fornecem informações tanto de dispositivos físicos quanto de partições. As saídas (métricas) que esse bloco apresenta são mostrados na Tabela 4.

Tabela 4: exemplo de `iostat` - utilização de dispositivo

Métrica	Descrição
`Device`	O nome do dispositivo físico ou partição presente no `/dev`.
`tps`	Indica o número de transferências por segundo que foi necessário executar no dispositivo. A transferência será uma requisição de I/O nesse dispositivo. Múltiplas requisições lógicas podem ser combinadas em apenas uma requisição de I/O no dispositivo.

Blk_read/s	Indica o montante de leitura de dados de um dispositivo expresso em número de blocos de 512 bytes por segundo.
Blk_wrtn/s	Indica o montante de gravação de dados em um dispositivo, expresso em números de blocos de 512 bytes por segundo.
Blk_read	Total de número de blocos lidos.
Blk_wrtn	Total de número de blocos gravados.
rrqm/s	Número de requisições de leituras mescladas por segundo. Que foram enfileiradas no dispositivo.
wrqm/s	Número de requisições de gravações mescladas por segundo que foram enfileiradas no dispositivo.
r/s	Número de requisições de leituras enviadas por segundo para o dispositivo.
w/s	Número de requisições de gravações enviadas por segundo para o dispositivo.
rsec/s	Número de setores lidos de um dispositivo por segundo.
wsec/s	Número de setores gravados no dispositivo por segundo.
avgrq-sz	Média do tamanho (em setores) das requisições enviadas a um dispositivo.
avgqu-sz	Média do tamanho da fila das requisições enviadas a um dispositivo.
await	Média do tempo (em milissegundos) para uma requisição de I/O ser atendida em um dispositivo. Inclui o tempo gasto pela requisição na fila e o tempo gasto para prover a informação.
r_await	Média do tempo (em milissegundos) para uma requisição de leitura ser atendida em um dispositivo. Isso inclui o tempo gasto pela requisição na fila e o tempo gasto para prover a informação.
w_await	Média do tempo (em milissegundos) para uma requisição de gravação ser atendida em um dispositivo. Inclui o tempo gasto pela requisição na fila e o tempo gasto para prover a informação.
%util	Percentual de tempo da CPU durante o qual foram feitas requisições de I/O ao dispositivo. Indica a taxa de utilização do

	dispositivo e moistra que o mesmo está saturado quando %util se aproxima de 100%.

A Erro: Origem da referência não encontrada apresenta as métricas de I/O de um dispositivo de disco rígido. Por essas métricas, visualiza-se a quantidade de blocos que um dispositivo precisou ler e gravar. Nesse exemplo, ocorreram mais leituras (`Blk_read`) no dispositivo do que gravação (`Blk_wrtn`) para essa interação do `iostat`.

A seguir, a Tabela 5 apresenta as opções de paramêtros do `iostat`.

Tabela 5: exemplo de `iostat` - parâmetros

Opção	Descrição
`-c`	Apresenta o relatório de utilização de CPU.
`-d`	Apresenta o relatório de utilização de dispositivo.
`-h`	Apresenta informações sobre a utilização de diretórios NFS. Para a leitura tradicional, utilizar a opção `-n`.
`-k`	Apresenta os números em KB por segundo, em vez de blocos por segundo. Opção válida apenas para kernel 2.4 ou posterior.
`-N`	Apresenta o nome do mapeamento do dispositivo registrado para qualquer dispositivo de mapeamento. Útil para visualizar estatísticas de LVM2.
`-n`	Apresenta o relatório de Network Filesystem (NFS). Opção válida apenas para kernel 2.6.17 ou posterior.
`-p [ {device | ALL }]`	A opção `-p` apresenta estatísticas para dispositivos de bloco e todas as suas partições que estão sendo usadas pelo sistema. Se o nome do dispositivo for informado, a estatística será visualizada. Opção válida apenas para kernel 2.5 ou posterior.
`-t`	Apresenta o horário em que cada relatório foi apresentado. O

	formato do timestamp vai depender do valor da variável de ambiente `S_TIME_FORMAT`.
`-V`	Imprime a versão do `iostat`.
`-x`	Apresenta o relatório de estatísticas estendidas. Essa opção funciona com o *kernel* 2.5 ou posterior. Pode funcionar com versões mais antigas do kernel (2.4 por exemplo), desde que informações estatísticas estendidas estejam disponíveis em /proc/partitions (o kernel precisará ser adaptado para isso).

3.1.3 `pmap`: instrumentação de memória por processo

O aplicativo `pmap` monitora o consumo de memória por processo. Com esse recurso, é possível fazer a leitura do consumo/utilização da memória por um determinado processo e seus respectivos subprocessos. O `pmap` basicamente coleta informações do `/proc` (`/proc/pid/maps` e `/proc/pid/smaps`), separando as métricas: endereço inicial do processo, tamanho, RSS (tamanho do mapeamento da memória física, incluindo a memória compartilhada – *resident set size*), PSS (tamanho dos processos utilizados na memória compartilhada – *proportional set size*), total de páginas *dirty*, as permissões do processo e o nome do processo em si.

Tabela 6: Exemplo de saída do comando `pmap`

28686: lame

START	SIZE	RSS	PSS	DIRTY	SWAP	PERM	MAPPING
0000000000400000	380K	260K	260K	0K	0K	r-xp	/usr/local/bin/lame
000000000065f000	4K	4K	4K	4K	0K	r--p	/usr/local/bin/lame
0000000000660000	4K	4K	4K	4K	0K	rw-p	/usr/local/bin/lame
0000000000661000	340K	200K	200K	200K	0K	rw-p	[heap]
00007fd066ad1000	1340K	372K	10K	0K	0K	r-xp	/lib64/libc-2.9.so
00007fd066c20000	2048K	0K	0K	0K	0K	---p	/lib64/libc-2.9.so
00007fd066e20000	16K	16K	16K	16K	0K	r--p	/lib64/libc-2.9.so
00007fd066e24000	4K	4K	4K	4K	0K	rw-p	/lib64/libc-2.9.so
00007fd066e25000	20K	16K	16K	16K	0K	rw-p	[anon]
00007fd066e2a000	340K	84K	38K	0K	0K	r-xp	/lib64/libm-2.9.so
00007fd066e7f000	2044K	0K	0K	0K	0K	---p	/lib64/libm-2.9.so
00007fd06707e000	4K	4K	4K	4K	0K	r--p	/lib64/libm-2.9.so
00007fd06707f000	4K	4K	4K	4K	0K	rw-p	/lib64/libm-2.9.so
00007fd067080000	120K	100K	1K	0K	0K	r-xp	/lib64/ld-2.9.so
00007fd06723a000	288K	144K	144K	144K	0K	rw-p	[anon]
00007fd067299000	16K	16K	16K	16K	0K	rw-p	[anon]
00007fd06729d000	4K	4K	4K	4K	0K	r--p	/lib64/ld-2.9.so
00007fd06729e000	4K	4K	4K	4K	0K	rw-p	/lib64/ld-2.9.so
00007fffdbcde000	1168K	900K	900K	900K	0K	rw-p	[stack]
00007fffdbec1000	4K	4K	0K	0K	0K	r-xp	[vdso]
ffffffffff600000	4K	0K	0K	0K	0K	r-xp	[vsyscall]
Total:	8156K	2140K	1629K	1320K	0K		

1848K writable-private, 6308K readonly-private, 0K shared, and 2140K referenced

A Tabela 6 evidencia um exemplo de instrumentação com o comando `pmap`, que apresenta o total de leitura (*readonly-private*) e gravação (*writable-private*) na memória durante a execução do aplicativo. Com esse dado, calcula-se o consumo de memória nesse processo, dado utilizado posteriormente para classificar em qual taxonomia a aplicação se encontra em termos de consumo de memória. No exemplo da Tabela 6, houve um consumo de 8.156 K de memória desde que o programa `lame` foi disparado.

A seguir, a Tabela 7 apresenta as opções de paramêtros do `pmap`.

Tabela 7: comando `pmap` - parâmetros

Opção	Descrição
`-d, --device`	Apresenta os dados estatísticos do dispositivo.
`-q, --quiet`	Esconde o cabeçalho com as estatísticas de memória.
`-h, --help`	Apresenta o menu de ajuda do `pmap`.
`-v, --version`	Imprime as informações da versão do `pmap`.

3.2 CARACTERIZAÇÃO DE APLICAÇÕES PARA COMPUTAÇÃO EM NUVENS

A caracterização de aplicações para computação em nuvens é fundamental para identificar e classificar programas que possuem as mesmas características, e assim definir configurações de MVs ideais e recomendadas para otimizar a alocação de recursos computacionais. Na perspectiva do provedor da nuvem, pode haver um estudo direcionado por aplicação, definindo grupos de aplicações por consumo. Sendo assim, o provedor tem um controle maior sobre a sua infraestrutura e permite dimensionar os equipamentos disponíveis na sua nuvem. Uma vez identificadas as características das aplicações, pode-se criar grupos (taxonomias) e definir possíveis comportamentos, antes mesmo de implementar a aplicação para o usuário final.

3.2.1 Aplicações do estudo inicial

Nesta dissertação, foram executadas aplicações distribuídas e transacionais, com o intuito de identificar as características destes dois tipos distintos de aplicações. No primeiro, espera-se encontrar uma elevada utilização de CPU e rede, pois o modelo dos programas indica estas duas características. Já nos transacionais, é aguardado um elevado consumo de disco, e uso moderado de CPU, dependendo do cenário a ser avaliado.

3.2.1.1 Aplicações distribuídas

As aplicações distribuídas são técnicas de desenvolvimento de sistemas usadas para

particionar o processamento de programas em diferentes computadores, criando, assim, um *cluster* fracamente acoplado. Uma forma de aplicar programação distribuída é por meio do MPICH2 (MPI) (GRABNER, 2004)(GROPP, 2002)(SNIR, 1995); ele faz com que a aplicação possa ser processada em nós (*hosts*) do MPI. O MPICH2 tem um ambiente de desenvolvimento interessante, que integra diferentes linguagens e outras bibliotecas, como `C` e `Fortran`. Essa flexibilidade permite que o MPICH2 seja uma forma interessante de avaliar o MPI, pois possibilita que a aplicação seja executada paralelamente, por meio de diversas ferramentas, compartilhando hardware e rodando as aplicações em ambientes HPC. Em termos de utilização dessas aplicações distribuídas, é possível desenvolver diferentes algoritmos para analisar o comportamento de I/O de rede, de utilização de CPU, memória e também de I/O de disco rígido.

Para coletar e analisar os dados do programa distribuído, é necessário ter uma ferramenta para coletar as métricas, como, por exemplo, tempos de execução e mops (*million operations per second*). O NPB (*NAS Parallel Benchmark*) (BAILEY, 1993) foi desenvolvido pela *Nasa Ames Research Center* para avaliar o desempenho das aplicações distribuídas e paralelas da própria corporação. NPB é um conjunto de programas executados em paralelo com a aplicação MPI ou OpenMP para, enfim, coletar as informações das métricas requeridas para uma análise da aplicação distribuída ou paralela. A principal proposta ao usar o NPB seria buscar parâmetros para executar cenários de avaliações em supercomputadores, *clusters* ou computadores unitários, executando o programa em paralelo (multiprocessadores) ou de modo distribuído.

Somente depois do surgimento do NPB 2.0 que o MPI teve o suporte para coletar informações de suas aplicações. Posteriormente, o NPB evoluiu para uma versão mais estável e a versão 3.3 foi lançada, adicionando suporte a versões de linguagens paralelas, como OpenMP, High Performance Fortran (HPF) e Java.

Na primeira versão, o NPB 1.0, três categorias foram criadas: classe W (para máquinas *workstations*), classe A (para computadores de médio porte) e classe B (para supercomputadores). Após o lançamento da versão 2.2, a classe C foi disponibilizada

(com o suporte ao MPI). Nessa versão, havia oito *benchmarks* possíveis para executar e coletar informações de aplicações distribuídas: *embarrassingly parallel* (EP), *multigrid calculation* (MG), *conjugate gradient* (CG), *fourier transform* (FT), *integer sort* (IS), *lower and upper* (LU), *scalar and pentadiagonal* (SP) e *block tridiagonal* (BT). A Tabela 8 apresenta as descrições de cada benchmark e suas características.

Tabela 8: NPB benchmarks (EL-GHAZAWI, 2002)

Benchmark	Descrição
BT	*Block tridiagonal benchmark* é uma aplicação de simulação CFD que usa um algoritmo implícito para resolver em 3D equações de *Navier Stokes*. A solução de diferenças finitas para o problema é baseada em uma ADI (*Alternating Direction Implicit*), que fatora a aproximação de blocos 5x5, que, por sua vez, são resolvidos sequencialmente ao longo de cada dimensão.
SP	*Scalar and pentadiagonal benchmark* é uma aplicação de simulação CFD estruturada do benchmark BT. A solução para o problema de diferenças finitas é baseada em uma *Beam-Warming* 12, que fatora a aproximação que separa as dimensões X, Y e Z. O sistema resultante tem bandas escalares pentadiagonais de equações lineares resolvidas sequencialmente ao longo de cada dimensão.
LU	*Lower and upper benchmark* é uma aplicação de simulação CFD que usa o método *symmetric successive over-relaxation* (SSOR) para resolver um sistema diagonal de sete blocos. Ela resulta das diferenças finitas e das discretizações das equações de Navier-Stokes em 3-D, dividindo-as em blocos de sistemas triangulares inferiores e superiores. O *benchmark* LU realiza um grande número de pequenas comunicações (cinco palavras) cada.
EP	O *embarrassingly parallel benchmark* pode ser executado em qualquer quantidade de processadores com pouca comunicação. Ele estima os limites superiores atingíveis para o desempenho do ponto

	flutuante de um computador paralelo. Esse *benchmark* gera pares aleatórios de *Gauss* de acordo com um esquema específico e tabula o número de pares de anéis sucessivos.
MG	*MultiGrid calculation benchmark* usa o método *multigrid* V-ciclo para calcular a solução da equação de *Poisson* em 3-D escalar. Ele executa as comunicações de curto e longo alcances, altamente estruturadas.
CG	*Conjugate gradient benchmark* calcula uma aproximação para o menor valor próprio da definição da matriz simétrica positiva. Esse recurso não estruturado requer uma rede de comunicações que demanda cálculos irregulares de longo alcance.
FT	*Fourier transform benchmark* resolve uma equação diferencial 3D parcial usando um método baseado em FFT espectral, exigindo também a comunicação de longo alcance. FT executa três unidimensionais (1-D) FFTs, um para cada dimensão.
IS	*Integer Sort benchmark* é um programa paralelo de ordenação baseado no algoritmo *bucket sort*, que exige comunicação intensa entre os processos paralelos.

3.2.1.2 Aplicações transacionais

As aplicações transacionais, por sua vez, utilizam bancos de dados relacionais para publicar e executar consultas em um sistema gerenciador de banco de dados (SGBD). Todos os recursos disponíveis em um SGBD relacional devem ser avaliados e considerados na definição dos cenários a serem executados e analisados. Recursos como transações têm como objetivo manter a integridade dos dados, gravando os registros nas tabelas somente no caso de sucesso da inclusão nas tabelas principais e auxiliares. Com isso, as informações ficam todas relacionadas por meio de chaves estrangeiras ou relacionamentos (*join*) nas consultas da linguagem SQL. As aplicações transacionais têm uma utilização intensa de HD, tanto para leitura como para gravação, porque ao inserir um registro no banco de dados, ocorre um processamento de

inserção de registro por meio da linguagem SQL, que, de uma estrutura lógica, faz o depósito em dispositivos de armazenamento de dados: *storage*, HD, SSD, entre outros. Cada dispositivo possui uma característica peculiar para o tratamento de leitura ou leitura e gravação. Para todos os periféricos, pode-se utilizar diferentes tipos de sistemas de arquivos, com alguma otimização na gravação do registro.

Para avaliar aplicações transacionais, foi utilizado o TPC-H, que permite coletar e analisar desempenho. A TPC é uma organização sem fins lucrativos criada para definir *benchmarks* de processamentos de transações em bancos de dados transacionais cujo intuito consiste em divulgar dados de desempenho obtidos, providenciar um comparativo entre outros concorrentes e ter um parâmetro. Na metodologia TPC-H, é possível fazer o download do aplicativo DBGEN, sendo permitido, portanto, utilizá-lo para gerar o banco de dados e a quantidade de registros para a execução das consultas criadas pelo QGEN. A princípio, o DBGEN gera um arquivo de texto com a quantidade de registros necessária para o tamanho do banco de dados solicitado na execução do DBGEN, cujo parâmetro é o próprio tamanho do banco de dados.

A Tabela 9 apresenta os principais parâmetros do comando `dbgen` depois de compilado:

Tabela 9: Principais parâmetros do `dbgen`

Parâmetros do `dbgen`	Descrição
`-h`	Apresenta a mensagem de ajuda do comando
`-f`	Caso existam os arquivos de saída do comando, estes são sobrescritos
`-F`	Saída do comando é enviada para arquivos. Esse parâmetro por padrão é habilitado
`-D`	Inclusão dos registros diretamente no banco de dados. É preciso definir o arquivo `load_stub.c`
`-s`	Escala numérica para a geração dos dados. Escala 1 representa ~1Gb de dados
`-T`	Geração de um dado específico de uma das tabelas do TPC-H:

	p → part e partsupp c → customer s → supplier o → orders/lineitem n → nation r → region l → code (é o mesmo que n e r) O → orders L → lineitem P → part S → partsupp

3.2.2 Primeiros estudos sobre a caracterização de aplicações

Em um primeiro estudo, objetivou-se apresentar uma avaliação da influência de alocação de recursos computacionais em aplicações com processamentos paralelos e transacionais. Com esse estudo preliminar, pôde-se obter alguns fatores importantes para a caracterização de consumo de recursos de acordo com os cenários propostos no experimento. E também foi possível identificar a configuração de CPU e disco (HD) nas MVs de acordo com a característica da aplicação.

Nos experimentos, foram executados programas paralelos e distribuídos, e programas transacionais em diferentes configurações de MVs.

Para os cenários da abordagem distribuída (Tabela 10), foi desenvolvido um algoritmo de multiplicação de matrizes em MPI, para executar uma atividade intensa de CPU nos nós do cluster MPI. Assim, foram utilizados os métodos de distribuição de tarefa via rede (troca de mensagens) para avaliar a influência da rede no processamento do programa. Para isso, foram criadas MVs para os cenários 1, 2 e 3 (Tabela 10). Outra análise experimentada foi utilizar a configuração multiprocessamento, para verificar a influência do acoplamento em uma MV. Portanto, foi criada uma MV com 2, 3 e 4 vcpus, conforme os cenários 4, 5 e 6 (Tabela 10).

Tabela 10: Cenários dos experimentos iniciais da dissertação

Cenário	Nome do cenário	Descrição	Configuração
1	1 proc 4 MV	4 MVs com 1 processador cada	2 MVs em cada nó
2	1 proc 3 MV	3 MVs com 1 processador cada	2 MV no nó principal e 1 MV no nó secundário
3	2 procs 2 MV	2 MVs com 2 processadores cada	1 MV em cada nó
4	2 procs 1 MV	1 MVs com 2 processadores	1 MV no nó principal
5	3 procs 1 MV	1 MVs com 3 processadores	1 MV no nó principal
6	4 procs 1 MV	1 MVs com 4 processadores	1 MV no nó principal

Na Tabela 10, são apresentados os cenários aos quais a aplicação MPI foi submetida. As seis situações tiveram múltiplos processadores alocados, porém a forma de configuração foi distribuída entre as MVs nos nós primário e secundário. Antes da execução propriamente dita, foram compilados os programas MPI (multiplicação de matrizes) e gerados os executáveis do benchmark NPB NAS, para suportarem 2, 4, 8, 16 e 32 processos de coleta de dados para cada execução do MPI. Para a execução do MPI, utilizou-se o comando mpiexec, por meio do qual se pode executar efetivamente o programa, passando a quantidade de processos pela opção -np #.

Tabela 11: Exemplo de execução do mpiexec com NPB

```
mpiexec -np 2 ./NPB3.3/NPB3.3-MPI/bin/is.C.2 ./m_matriz >
results/results_2_1 < /dev/null
```

Na Tabela 11, é apresentado um exemplo de execução do `mpiexec` com dois processos; o comando executa em paralelo o NPB IS para 2 processos, e o programa m_matriz (executável do algoritmo de multiplicação de matrizes). Os resultados serão gravados no arquivo `results/results_2_1`. Caso ocorra alguma mensagem de erro, o comando indica o envio para o `/dev/null`, para não interromper o processamento do programa.

Posteriormente, foi criado um *script bash* (APÊNDICE J) com vinte iterações para

cada número de processo (4, 8, 16 e 32), bem como passou-se o arquivo compilado do NPB IS de acordo com a quantidade de processos do MPI. Após o término da execução do script bash do primeiro cenário, foi reconfigurado (reiniciada a MV, para garantir que a memória fosse limpa e desligada, caso não fosse parte do cenário) o ambiente para o segundo cenário, deixando ligadas apenas as MVs que fariam parte do ambiente, e reexecutado o script com as mesmas quantidades de processos com vinte iterações cada. Sucessivamente, executou-se cada cenário da Tabela 10.

Já no experimento transacional, o objetivo foi avaliar a influência de uma aplicação com alta utilização de disco rígido (HD) no tratamento de leitura de disco por intermédio do *benchmark* TPC-H e SGBD IBM DB2. Primeiramente, compilou-se o TPC-H (`dbgen`) para a geração de dados, utilizando-se as opções: `dbgen -s 2 -f`. O comando aplicou a escala de 2, efetuando a reescrita, caso o arquivo de saída já existisse. Portanto, foram gerados ~2 Gb (~7 milhões de registros) de dados distribuídos nas 8 tabelas do TPC-H. O valor de escala 2 foi um número aleatório (o dobro do valor padrão do comando), e como os resultados de desempenho foram satisfatórios, não foi necessário fazer o incremento dos números de registros. Com os dados gerados e importados para o banco de dados relacional DB2, chamado de TPCH2, iniciaram-se os experimentos desse teste:

Tabela 12: Cenários do TPC-H

Cenário	Nome do cenário	Tipo de sistema de arquivo
1	Local	Ext3
2	NFS com o dom0	NFS com ext3
3	NFS com servidor NFS remoto	NFS com ext3

Os cenários da Tabela 12 foram definidos para avaliar o desempenho de uma aplicação que utiliza leitura de disco rígido com intensidade, verificando se a tecnologia associada penaliza o desempenho de uma aplicação que faz uso do NFS. Na teoria, o cenário 1 teria um desempenho melhor pelo fato de a aplicação estar sendo executada na mesma região do sistema de arquivos, não havendo intermediários para atrasar o

processamento. Já no caso do NFS com o dom0, entende-se que este possui um algoritmo de otimização para melhorar a comunicação da rede via dom0, portanto, utiliza a memória compartilhada para fazer comunicação com a rede. Sendo assim, espera-se um desempenho que possa justificar a adoção desse meio para comunicar aplicações que requeiram a rede. Por fim, NFS com um servidor NFS remoto seria o pior cenário, pois este precisa se comunicar por meio de rede física, havendo efetivamente comunicação via TCP-IP.

Após a configuração da MV contendo um diretório (FS ext3) configurado, um NFS com o dom0 e outro diretório com NFS com uma máquina remota, foi executado um *script bash* (APÊNDICE K) com um laço de vinte iterações, para as consultas complexas (Tabela 14) e *table scan* (Tabela 13) - primeiramente no cenário local, depois no NFS com o dom0 e, por fim, no NFS com a máquina remota.

Tabela 13: Consulta *table scan*

```
select max(l_discount) from lineitem where l_shipdate<'1992-
07-01';
```

Tabela 14: Consulta complexa

```
select c_name, c_custkey, o_orderkey, o_orderdate,
o_totalprice, sum(l_quantity)
from customer, orders, lineitem
where o_orderkey in (
    select l_orderkey
    from lineitem
    group by l_orderkey
    having sum(l_quantity) > 312
)
and c_custkey = o_custkey and o_orderkey = l_orderkey
group by c_name, c_custkey, o_orderkey, o_orderdate,
o_totalprice
order by o_totalprice desc, o_orderdate;
```

3.2.3 Ambiente computacional

Na Tabela 15, são apresentadas as configurações das duas máquinas reais e a configuração básica das MVs utilizadas no experimento. Tendo em vista o fato de o nó principal ter sido instalado, o programa de computação em nuvens privado (OpenNebula) não precisa do nó secundário, pois o OpenNebula requer apenas o MMV Xen para gerenciar as MVs. Por isso, no caso do Xen, foi preciso instalar em ambos os equipamentos.

Tabela 15: Configuração das máquinas reais e virtuais

Nó	Hardware		Software	
Principal	Processador	AMD Phenon 9600 Quad-core 2.3 GHz	Sistema operacional	Opensuse 11.1 64 bits
	Memória	4 Gb	Kernel	2.6.27.56 -0.1-xen
	Disco	1 Tb	Xen	3.3
	Rede	1 interface 1GbE Full-duplex (rede interna) 1 interface 100 MbE Full-duplex (rede do laboratório LAHPC)	OpenNebula	1.4.0
Secundário	Processador	AMD Phenon 9650 quad-core 2.3 GHz	Sistema operacional	Opensuse 11.1 64 bits
	Memória	4 Gb	Kernel	2.6.27.56 -0.1-xen
	Disco	1 Tb	Xen	3.3
	Rede	1 interface 1GbE Full-duplex (rede interna) 1 interface 100 MbE		

		Full-duplex (rede do laboratório LAHPC)		
MVs	Processador	[Depende do cenário]	Sistema operacional	Opensuse 11.1 64 bits
	Memória	[Depende do cenário]	Kernel	2.6.27.56 -0.1-xen
	Disco	60 Gb	TPC-H	2.12.0
	Rede	1 interface 1GbE full-duplex (rede interna)	NAS NPB	3.3
			IBM DB2 Express-C	9.7
			PBZIP2	1.1.3
			LAME	3.98.4
			Systat	8.1.5
			Pmap	3.2.7

3.2.4 Tipos de aplicações

As subseções seguintes descrevem os tipos de aplicações a serem exploradas e os experimentos usados para a instrumentação de um aplicativo. Uma vez detalhada cada característica da aplicação, é possível criar as regras de classificação para o método, com o intuito de possibilitar a incorporação desse mecanismo de classificação na metodologia proposta nesta dissertação.

3.2.4.1 CPU *bound*

Para os aplicativos que utilizam um percentual significativo de CPU, é indicado configurar a MV com multiprocessadores na mesma máquina virtual. Porém, a aplicação demanda suporte a programas paralelos ou a alguma forma de particioná-la em diferentes processos que possam ser alocados em diferentes processadores. Caso as aplicações tenham suporte a programas paralelos e distribuídos, recomenda-se

utilizar mais de uma MV com multiprocessadores alocados. O problema de utilizar as MVs em ambiente distribuído é haver atraso na comunicação entre os processos, porém se ganha em escalabilidade e flexibilidade de um ambiente computacional.

Os aplicativos que trabalham com utilização de CPU moderada dedicam recursos de processador mediano, porque este não requer tanta capacidade de processamento, como no caso anterior. Para isso, a tendência é utilizar a abordagem de ambientes distribuídos, para aproveitar a estrutura de acoplamento fraco, possibilitando a utilização de diferentes servidores, com diversos MVs.

Por fim, quanto aos produtos computacionais que não requerem tanto processamento de CPU, indica-se configurar a MV com um processador, ou por intermédio de MVs distribuídas, porém com capacidade de processamento reduzida, pois a necessidade é baixa.

3.2.4.2 I/O de disco e memória

Com relação a aplicativos com utilização de disco elevada, o I/O é uma forma de analisar o quanto de *workload* está sendo requisitado para ler ou gravar dados. Essa métrica de I/O pode indicar uma série de estatísticas de tráfego de dados, espera (*delay*), quantidade de blocos processados, consumo de CPU, etc. Além disso, é possível separar as métricas de I/O de leitura e gravação; elas possibilitam uma análise detalhada independentemente. O principal motivo de se analisar separadamente é que alguns periféricos possuem um tratamento mais eficiente para situações de leitura de dados (*Solid State Drive* – SSD).

Portanto, para aplicações com consumo elevado de leitura em disco, e para as quais há pouca ou moderada gravação, pode-se indicar a utilização de dispositivos SSDs ou periféricos locais na MV. Se a aplicação possuir consumo mediano de leitura e gravação no dispositivo, recomenda-se utilizar um *Network File System* (NFS) ou qualquer outro tipo de sistema de arquivo que utilize a rede para sincronizar dados com o equipamento hospedeiro (dom0), pois o Xen possui a otimização que sincroniza os dados por intermédio da memória compartilhada do dom0. Todavia, no caso de baixa leitura e baixa gravação, pode-se utilizar o NFS com outras máquinas físicas, pois a necessidade de desempenho é baixa, o que possibilita o uso de mecanismos de NFS.

Além do mais, o NFS permite uma série de flexibilidades na migração ou transmissão de dados de uma forma automática. Essa abordagem do NFS pode apresentar dificuldades, pois se trata de um método que requer utilização intensa da rede para sincronizar os *File Systems* (FS) e podem ocorrer atrasos na leitura ou gravação.

As métricas de memória podem ser classificadas em dois grupos. A primeira é para os aplicativos que requerem utilização intensa de memória. Para esse grupo, recomenda-se utilizar memória local na MV. Para os aplicativos com utilização de memória mediana ou baixa, pode-se alocar memória distribuída por meio de diferentes MVs e os processos, por sua vez, fazem o processamento de modo paralelo e distribuído. Caso não seja possível utilizar o método distribuído, aloca-se memória local na MV.

3.2.4.3 I/O de rede

Aplicativos que possuem uma intensa utilização de rede necessitam de banda de rede para enviar e receber uma quantidade de dados elevada. Para os aplicativos que estão nessas taxonomias, pode-se utilizar uma tecnologia de 10 Giga bit Ethernet (GbE), padrão IEEE 802.3ab, para transmitir dados entre os nós do *cluster*. Essa tecnologia de 10 GbE inicialmente foi desenvolvida em cabos de fibra ótica, porém há informações de que cabos de cobre e equipamentos de rede suportam 10 GbE com algumas limitações, o que não ocorre com a fibra ótica. Contudo, o custo de implementação ainda é um obstáculo, pelos altos preços dos equipamentos de redes e cabeamentos para a tecnologia de fibra ótica.

Com relação aos programas que utilizam a rede moderadamente, pode-se configurar uma rede de 1 GbE, praticamente um padrão de mercado, pois a maioria dos equipamentos de rede suporta essa tecnologia. Além disso, pelo fato de a utilização ser mediana, pode-se utilizar a otimização do Xen para transmitir uma informação pela rede das MVs por meio do dom0 (memória compartilhada). No entanto, quando a comunicação é externa à MV, o dom0 não utiliza esses algoritmos de otimização; efetivamente, usam-se as camadas TCP/IP.

Para aplicações com baixa utilização de rede pode-se utilizar a comunicação

tradicional, sem qualquer otimização. Uma rede de 100 MbE é o suficiente para suportar a demanda das solicitações de envio e recebimento de informações de um nó para outro.

3.2.5 Resultados do estudo inicial

Para a fase inicial do estudo, foram coletadas informações dos cenários de CPU BOUND e I/O de disco, com os cenários de aplicações paralelas Tabela 10 e transacionais na Tabela 12.

Os resultados dos cenários da Tabela 10 e Tabela 12 são apresentados nos itens 3.2.5.1 até 3.2.5.4.

3.2.5.1 Resultados de CPU *bound* – abordagem de multiprocessadores

Nesta subseção, serão analisados os resultados de uma MV com mais de uma vcpu atribuída para o processamento da aplicação MPI. Esta abordagem de multiprocessador tem a intenção de executar aplicações científicas (Multiplicação de matrizes), apenas variando a quantidade de "vcpus" com 2, 3 e 4 vcpus (Tabela 10). Em geral, essa configuração possibilita avaliar o comportamento do algoritmo, dividindo o processo MPI e criando threads para serem alocadas em cada vcpu da MV. Sendo assim, para cada thread, um rank do MPI foi atribuído, e iniciou-se a comunicação por meio de memória compartilhada, pois o MPI otimiza esse meio de comunicação, em vez de utilizar a infraestrutura de rede TCP/IP. Além dessa otimização, ao atribuir mais vcpus para processar esse programa MPI, há maior divisão em threads, ou seja, menos workload por processador.

Tabela 16: Resultados de CPU *bound* – abordagem de multiprocessador (unidade em segundos)

Número de processos			4	8	16	32
Cenário 4	2 proc 1 VM	Avg	15,6	17,3	20,5	26,6
		Std Dev	0,11	0,11	0,1	0,37
Cenário 5	3 proc 1 VM	Avg	12,4	13,8	16,6	21,2
		Std Dev	0,29	0,2	0,91	0,36
Cenário 6	4 proc 1 VM	Avg	10,4	11,9	14,7	18,9
		Std Dev	0,11	0,15	0,21	0,45

A Tabela 16 apresenta os tempos de execução do algoritmo (multiplicação de matrizes) que foram necessários para fazer o processamento. Ao término de vinte iterações, o benchmark apresenta as medidas dos tempos de execução e mops, entre outras informações. Esse método foi aplicado em MVs com 2, 3 e 4 vcpus e quantidade de processos de MPIs com 4, 8, 16 e 32. Essa configuração viabilizou a análise das execuções, utilizando o multiprocessador para distribuir os processos MPI na MV. Para as quantidades de processos maiores do que a quantidade de vcpus atribuídas à MV, o MPI precisou executar os processos simultaneamente, havendo concorrência de processamento. Por este motivo, ao aumentar a quantidade de processos, o tempo de execução também irá aumentar, pois a vcpu terá mais processos. Sendo assim, foi executado o experimento nestas condições para ser avaliada a degradação ao se aumentar a quantidade de processos. Após a execução dos cenários e da coleta das métricas, no cenário de 4 vcpus em uma MV, obteve-se um desempenho superior em relação às outras situações. Portanto, a teoria de que quanto mais processadores alocados em uma MV (para aplicações de multiprocessador), pode-se obter melhores desempenhos. Além disso, o MPI possui otimização de comunicação ao executar cenários com multiprocessador, nos quais a comunicação entre os processos de MPI ocorre por meio de memória compartilhada. Esse método faz com que os processos efetivem e transmitam o dado processado com desempenho, não havendo barreiras (*overhead*) para atrasar o retorno da informação.

3.2.5.2 Resultados de CPU *bound* – aplicação em ambiente distribuído

Esta análise, por sua vez, tem a intenção de avaliar o comportamento da aplicação MPI por meio de aplicações distribuídas, ou seja, a aplicação MPI foi implementada em duas, três e quatro MVs na mesma máquina hospedeira. Com isso, o método de comunicação entre as MVs se deu por intermédio da memória compartilhada, pois o Xen possui um método de otimização que utiliza o dom0 (CHERKASOVA, 2005) para encapsular a rede na memória compartilhada do dom0, obtendo uma comunicação mais eficiente.

Tabela 17: Resultados de CPU *bound* – abordagem de aplicação distribuída (unidade em segundos)

Número de processos			**4**	**8**	**16**	**32**
Cenário 1	1 proc 4 VM	Avg	32,2	43,3	52,7	75,7
		Std Dev	2,05	2,86	1,94	2,27
Cenário 2	1 proc 3 VM	Avg	40,3	46,7	52	82,1
		Std Dev	4,66	4,84	5,87	5,27
Cenário 3	2 proc 2 VM	Avg	14,5	16,7	21,2	29,9
		Std Dev	0,51	0,35	0,62	0,72

Ao analisar os resultados da Tabela 17, duas situações devem ser detalhadas. A primeira é que nos dois cenários com monoprocessador, mas com 3 e 4 MVs (Tabela 17), pode-se observar que o desempenho não foi satisfatório, porque o melhor tempo de execução foi de 32,2 segundos (cenário 1 - 1 processador e 4 VM - 4 processos), quase três vezes mais lento em relação ao caso oposto a esse cenário (Cenário 6 - 4 processadores 1 VM - 4 processos), que levou 10,4 segundos para processar (Tabela 16). A segunda situação, por outro lado, o cenário três, com duas vcpus e duas MVs, apresentou-se interessante no desempenho, pois ao se fazer uma combinação de cenários com multiprocessamento e processamento distribuído, pode-se aproveitar melhor os processadores na mesma MV, não havendo necessidade de comunicação intensa entre as MVs. Os resultados apresentados refletem o comportamento do

desempenho do algoritmo do benchmark IS (Tabela 8), pois a principal característica do IS é haver muito I/O de rede para enviar uma mensagem para cada nó rank do MPI. Se cada rank pegar dois processos e utilizar a memória compartilhada para comunicar-se entre os ranks, executará o algoritmo com mais eficiência.

3.2.5.3 Resultados de I/O de disco – aplicação transacional

Esta subseção tem o objetivo de avaliar o comportamento de aplicações transacionais em ambientes de MVs. As aplicações transacionais possuem características diferentes das aplicações analisadas anteriormente, tendo como principal característica a utilização intensa de disco (I/O) tanto para leitura quanto para gravação. Em certo momento, a leitura ocorrerá com maior quantidade e, em outros, a gravação será maior. Uma vez entendida essa característica de uma aplicação transacional, esta pode ser implementada por meio de bancos de dados (BD) relacionais, pois os SGBDs relacionais possuem suporte a transações e pode-se avaliar a influência da implementação de BDs em diferentes sistemas de arquivos (FS).

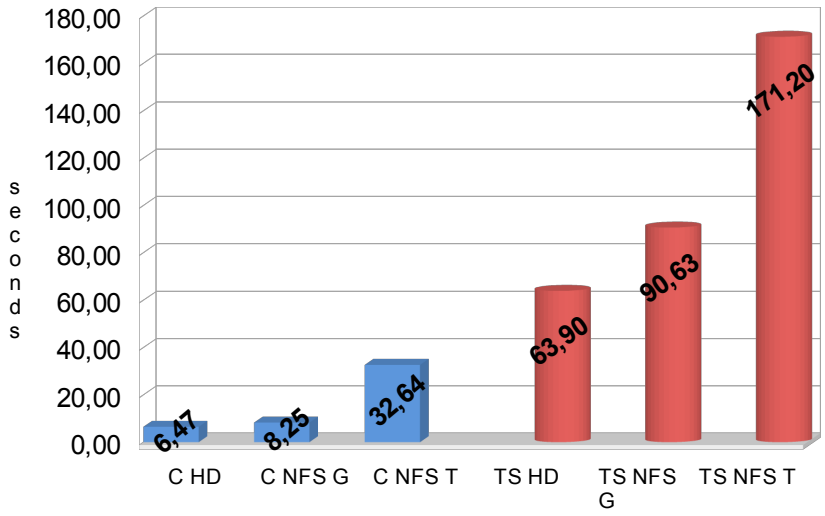

Figura 13: Abordagem de aplicações transacionais

A Figura 13 apresenta os tempos médios das execuções das consultas de tipo *table scan* e complexa em dois tipos de sistemas de arquivos, ext3 e NFS. Para este

experimento, foi executado em cada cenário, conforme a Tabela 18, vinte iterações para cada consulta (table scan e complexa). Com isso, pode-se avaliar o comportamento de cada consulta (leitura) em execução nos FS local, NFS com o dom0 e NFS remoto.

Tabela 18: Aplicações transacionais – FS utilizados

Legenda	Descrição
C HD	Consulta complexa em sistema de arquivos na MV (ext3).
C NFS G	Consulta complexa em sistema de arquivos NFS com a máquina hospedeira (NFS com o dom0).
C NFS T	Consulta complexa em sistema de arquivos NFS com outra máquina (NFS com uma máquina remota).
TS HD	Consulta *table scan* em sistema de arquivos na MV (ext3)
TS NFS G	Consulta *table scan* em sistema de arquivos NFS com a máquina hospedeira (NFS com o dom0).
TS NFS T	Consulta *table scan* em sistema de arquivos NFS com outro máquina. (NFS com uma máquina remota).

Como era esperado, os cenários "C HD" e "TS HD" obtiveram o melhor desempenho na execução das consultas complexa e table scan, porque ambos os cenários executaram consultas no banco de dados que estava armazenado fisicamente no sistema de arquivo (ext3) da MV. Portanto, não há intermediários para atrasar o processamento dessa consulta. Por outro lado, as consultas "C NFS G" e "TS NFS G" alcançaram resultados satisfatório, pois esses cenários obtiveram resultados próximos ao local. Contudo, o tempo de execução das consultas table scan e complexa sofreu um acréscimo de ~22% na consulta complexa, enquanto a consulta table scan teve um acréscimo de ~40% com relação ao cenário do banco de dados em um sistema de arquivos na própria MV e NFS com o dom0. Esse percentual de acréscimo muitas vezes é justificado em um ambiente computacional, pois possibilita mover uma infraestrutura entre MVs no mesmo dom0 ou servidores NFS remotos. Por fim, o último cenário, "C NFS T" e "TS NFS T", não teve um desempenho bom. A justificativa para

esse desempenho se dá por haver necessidade de comunicação com outra máquina real, demandando obrigatoriamente a comunicação TCP/IP. Além disso, existe o problema de sincronismo (NFS) do sistema de arquivos com a máquina servidora de NFS, havendo um overhead na leitura dos registros em um sistema de arquivos NFS.

3.2.5.4 Comparativo entre cenários com a mesma quantidade de vcpus

Esta subseção tem o objetivo de apresentar um comparativo entre os cenários que possuem a mesma quantidade de processadores alocados, independentemente da forma como foram implementados. Pode-se analisar os resultados de cada cenário contendo 3 e 4 vcpus atribuída nas MVs. Com isso, características sobre processamento paralelo e distribuído podem fornecer informações relevantes sobre a adoção da configuração em uma MV.

Tabela 19: Comparação dos cenários com 4 vcpus (unidade em segundos)

Número de processos			**4**	**8**	**16**	**32**
Cenário 1	1 proc 4 VM	Avg	32.2	43.3	52.7	75.7
		Std Dev	2.05	2.86	1.94	2.27
Cenário 3	2 proc 2 VM	Avg	14.5	16.7	21.2	29.9
		Std Dev	0.51	0.35	0.62	0.72
Cenário 6	4 proc 1 VM	Avg	10.4	11.9	14.7	18.9
		Std Dev	0.11	0.15	0.21	0.45

A Tabela 19 apresenta os cenários 1, 3 e 6 (vide Tabela 10) que possuem 4 vcpus na soma dos processadores alocados na (s) MV (s). Pode-se observar que os cenários que utilizaram a memória compartilhada para a comunicação entre os processos MPI obtiveram desempenhos semelhantes (3 e 6). Contudo, o cenário 1 não apresentou bom desempenho, pelo fato de haver 4 MVs na mesma máquina hospedeira. Portanto, a combinação de multiprocessamento (cenário 6) e distribuído (cenário 3) pode ser uma configuração indicada para conseguir resultados positivos em

termos de desempenho das aplicações.

Outra análise que pode ser feita é a dos cenários 2 e 5. Estes dois experimentos utilizaram a mesma quantidade de vcpus (três), porém uma configuração diferente.

Tabela 20: Comparação dos cenários com 3 vcpus (unidade em segundos)

Número de processos			4	8	16	32
Cenário 2	1 proc 3 VM	Avg	40.3	46.7	52	82.1
		Std Dev	4.66	4.84	5.87	5.27
Cenário 5	3 proc 1 VM	Avg	12.4	13.8	16.6	21.2
		Std Dev	0.29	0.2	0.91	0.36

Na Tabela 20, pode-se comprovar que a abordagem de multiprocessamento obteve um desempenho três vezes mais eficiente do que o cenário com 3 MVs. Com isso, recomenda-se utilizar multiprocessador em vez de distribuir o processamento em MVs diferentes, mesmo utilizando o dom0 para encapsular a comunicação via rede, pois haverá atraso na devolução do resultado por conta dessa comunicação.

3.2.6 Conclusão do estudo inicial

Na fase inicial deste estudo, objetivou-se a identificação da melhor configuração de MVs, de acordo com aplicações com diferentes características (MPI e TPC-H). Para essas aplicações, foram executados diferentes cenários no intuito de avaliar o desempenho de cada um e de tirar algumas conclusões do desempenho do programa de acordo com a configuração da MV. Por meio dos experimentos e dos resultados, concluiu-se que, para aplicações com suporte a processamento paralelo e distribuído, indica-se a combinação de MVs com mais de um processador e distribuição em diferentes MVs.

3.3 CARACTERÍSTICAS DE APLICAÇÕES

Nesta seção, a caracterização das aplicações possibilita uma análise prévia de uma determinada aplicação. Por meio de características específicas, pode-se classificar as

aplicações com uso da lógica Fuzzy. A partir desses recursos, pode-se coletar métricas de I/O, tempo e *workload* para identificar características ou particularidades de um programa ou aplicativo. As subseções a seguir apresentam o conceito de classificação Fuzzy e a definição de três tipos de aplicações.

3.4 METODOLOGIA PROPOSTA

A metodologia proposta nesta dissertação tem o objetivo de apresentar uma solução de caracterização de aplicações por intermédio de um plano de instrumentação e classificação de programas computacionais em ambientes de máquinas virtuais. Estes métodos possibilitam criar cenários e apresentar diferentes grupos para tipos de aplicações variadas. Sendo assim, este esquema pode ser incrementado conforme o aumento da quantidade de aplicações instrumentadas e inseridas aqui, permitindo que outras variáveis comparativas possam ser utilizadas para uma precisão maior na categorização dos grupos, indicando a configuração ideal ou recomendada para cada aplicação. A seguir, serão abordadas as fases que compõe a metodologia.

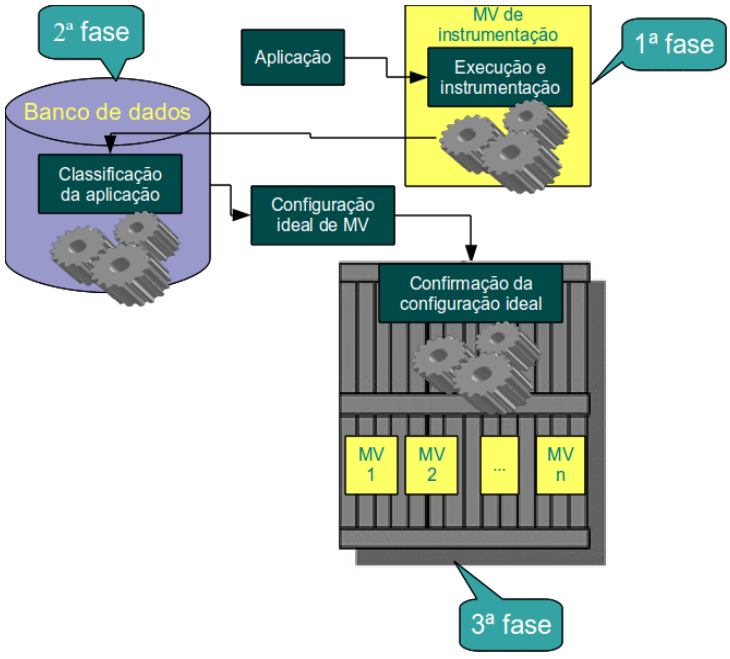

Figura 14: metodologia completa

3.4.1 Etapas da metodologia

Para um entendimento detalhado do plano proposto, a metodologia foi segmentada em três etapas. Cada etapa é descrita a seguir.

3.4.1.1 Primeira etapa

A etapa inicial deste trabalho foi a execução da aplicação em uma MV com configuração única (1 vcpu, 1.4 Gb de memória, 88 Gb de disco) com o intuito de executar a aplicação em um ambiente computacional único para classificar o programa utilizando o mesmo método e configuração alocado na MV. E em paralelo, os comandos de instrumentação para a coleta das métricas para os dispositivos a serem utilizados para a classificação de intensidade de consumo. Ao término da execução da aplicação, são obtidos um arquivo contendo o tempo que o programa levou para executar, e a quantidade de I/O (`iostat`) e de memória (`pmap`) utilizadas no processo da execução. Adicionalmente, para os cenários com NFS, dom0 e aplicação local na MV, foi acrescentada uma leitura do `iostat` e do `xentop`. O objetivo, ao coletar dados do `iostat` do dom0, é verificar a influência que este tem no resultado final do processamento do programa. Já o `xentop` apresenta o consumo de CPU entre o dom0 e a MV, sendo possível tirar algumas conclusões sobre como o dom0 está consumindo CPU durante o processamento do programa na MV. No cenário de NFS com uma máquina remota, o `iostat` e o `xentop` foram executados na máquina hospedeira (dom0), e adicionalmente foi processado o `iostat` no servidor de NFS remoto, fazendo a leitura dos I/Os do *File System* exportado do NFS. Quando o programa finaliza, os comandos de instrumentação são encerrados na MV, no dom0 e no NFS remoto.

3.4.1.2 Segunda etapa

Com os dados coletados, a segunda etapa refere-se ao processo de classificação da aplicação de acordo com as medidas extraídas da primeira etapa. Para que pudessem ser classificadas em taxonomias, foi necessário identificar um método para separar três grupos classificatórios por cada recurso computacional instrumentado na primeira

etapa. Sendo assim, para esta dissertação foi definido avaliações de recursos computacionais de CPU, memória, HD leitura e HD gravação. Para cada grupo, foi dividido em três categorias: utilização intensa (+), média e baixa (-). Uma vez categorizado, pode-se inserir as leituras no banco de dados que contém as instrumentações anteriores para enfim dividir entre todas as leituras, nos três grupos sugeridos nesta dissertação. Para este processo de divisão, utilizou-se a metodologia *Crisp* (*Fuzzy*) para dividir todos os registros de cada recurso computacional em três grupos conforme descritos na Tabela 21. Quanto maior a quantidade de registros instrumentados, melhor vai ser o método classificatório para indicar a configuração ideal para o processamento desta aplicação. Para esta etapa, foi desenvolvido um *script* (APÊNDICE A) e executado em todos os programas instrumentados e avaliados nesta dissertação, para que os bancos de dados pudessem ter histórico para as distribuições dos três tipos de aplicações (taxonomias). Por meio das tabelas `instrumentacao` e `metrica`, é possível obter informações referentes aos valores de CPU, memória, HDR e HDW. Nesta dissertação, não serão apresentadas métricas de rede, pelo fato de as métricas de CPU, memória e HD serem suficientes para justificar a metodologia de caracterização de aplicações.

Após completar a transação no banco de dados, pode-se executar a *view* (APÊNDICE D) `vw_metrica` pelo comando SQL (`select * from vw_metrica`) para apresentar o resultado atual das medidas, indicando qual taxonomia é adequada para cada registro da tabela `instrumentação`, relacionada com `metrica`. Os campos que serão apresentados na consulta serão: `avgmetrica` (média do valor da métrica), `desviopad` (desvio padrão dos valores da métrica), `dscmetrica` (descrições da métrica), `codexecucao` (código da execução da aplicação, atribuído na primeira etapa) e `grupo` (execução da função `F_CATEG_FUZZY`, (APÊNDICE C) que comporta a categoria em que o valor da métrica se encontra.

Tabela 21: Classificação *Fuzzy vs* recurso ideal/recomendado

Taxonomia	Descrição	Configuração ideal	Configuração recomendada	Métrica	Fórmula matemática [1]
CPU+	Utilização alta de CPU	Multiprocessador	Processamento Distribuído com multiprocessador	`iostat` → %user	$\left(\sum \%\text{user}\right)/n$
CPU	Utilização média de CPU	Multiprocessador	Processamento Distribuído	`iostat` → %user	
CPU-	Utilização baixa de CPU	Processamento Distribuído	Monoprocessador	`iostat` → %user	
MEM+	Utilização alta de memória	Memória local	Memória local	`pmap` → writable-private + readonly-private	$\left(\sum \left(\text{writable-private} + \text{readonly-private}\right)\right)/n$
MEM	Utilização média de memória	Memória local	Memória distribuída	`pmap` → writable-private + readonly-private	
MEM-	Utilização baixa de memória	Memória distribuída	Memória distribuída	`pmap` → writable-private + readonly-private	
HDR+	Utilização alta de leitura a HD	Local	SSD	`iostat` → r/s	$\left(\sum \text{r/s}\right)/n$
HDR	Utilização média de leitura a HD	NFS com o DOM0	NFS com o DOM0	`iostat` → r/s	
HDR-	Utilização baixa de leitura a HD	NFS remoto	NFS remoto	`iostat` → r/s	
HDW+	Utilização alta de gravação em HD	Local	Local	`iostat` → w/s	$\left(\sum \text{w/s}\right)/n$
HDW	Utilização média de gravação em HD	NFS com o DOM0	NFS com o DOM0	`iostat` → w/s	
HDW-	Utilização baixa de gravação em HD	NFS remoto	NFS remoto	`iostat` → w/s	

A Figura 15 ilustra a estrutura das tabelas do banco de dados utilizado para armazenar os registros das execuções. A tabela principal é a `instrumentacao`, que armazenará os dados de instrumentação, o valor da métrica, o código da métrica e o código da execução. Os códigos de métrica e execução são campos de relacionamento com as tabelas `metrica` e `execucao`. Por fim, a última tabela é apenas informativa e evidencia manualmente o código da máquina em que se está executando esse experimento. Para utilizar um novo equipamento ou MV, é necessário fazer a inclusão

[1] n = número de elementos da métrica

do registro previamente. Porém, essa informação será apenas para utilização futura, pois o usuário dessa metodologia possivelmente precisará saber os dados do equipamento de instrumentação que foi executado, incluindo configuração e especificação do sistema operacional.

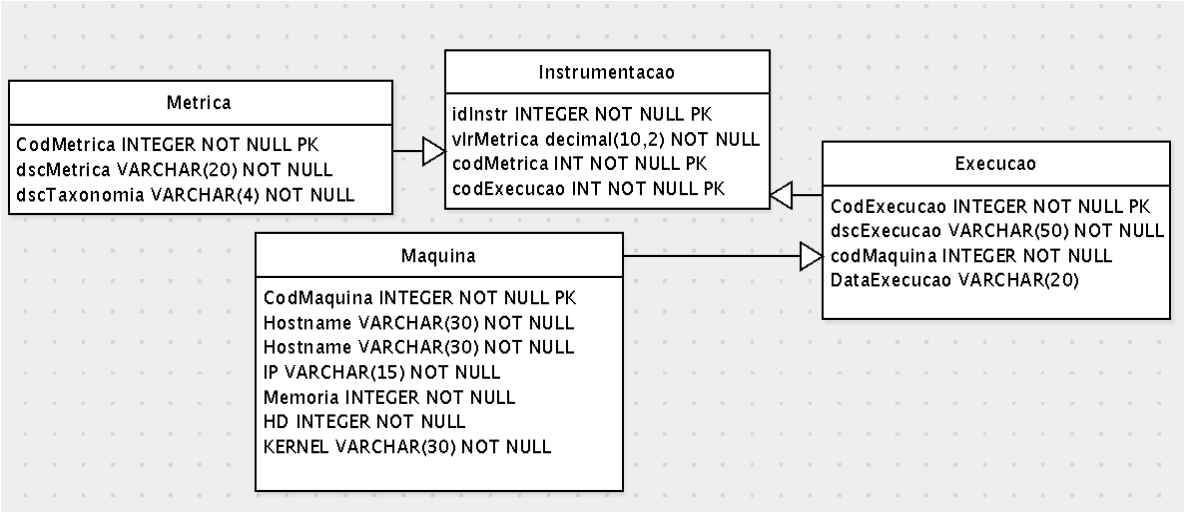

Figura 15: DER do banco de dados

3.4.1.3 Terceira etapa

A etapa final é a certificação de que o método apresenta a melhor configuração de acordo com um conjunto de aplicativos já instrumentados anteriormente. A segunda etapa provê como saída de resultado a configuração recomendada para executar a aplicação instrumentada de acordo com a classificação e característica de consumo do recurso computacional, comparada a outras aplicações instrumentadas anteriormente. Para isso, esta etapa objetivou-se a comprovar de que a aplicação obtém um desempenho satisfatório ao configurar a MV com a configuração sugerida. Para isso, as MVs foram configuradas nos cenários sugeridos pela Tabela 21, submetido a aplicação nas MVs e obtido as medidas de instrumentação para a comparação das médias de cada recurso computacional. Sendo assim, pode-se certificar de que a configuração sugerida pela segunda etapa é satisfatória.

3.5 TRABALHOS RELACIONADOS

A computação nas nuvens vem sendo alvo de diversas linhas de pesquisa, discussões e fóruns, pois se trata de um assunto entendido por muitos como uma nova tendência comercial e acadêmica para os próximos anos, ou seja, uma alternativa promissora. Por isso, muitas empresas têm investido nesse tipo de arquitetura computacional. Todavia, em termos de avaliação de desempenho, ainda é um assunto que precisa ser explorado, porque há bastante campo para ser observado.

Dentro do que foi estudado, os autores (ORDUNA, 2000) investigaram o comportamento de aplicações MPI utilizando a comunicação da rede e o MPI *cluster* como métodos para comunicar processos remotos. Eles utilizaram o MPICH e NPB *benchmarks* (CG, EP, IS, LU, MG e SP) com 8 e 64 processos. A conclusão do artigo foi que a comunicação, em geral, não é uniformemente distribuída entre os processos, com base nas tabelas e gráficos apresentados no trabalho. Para caracterizar os requisitos da comunicação, os autores aplicaram um aglomerado de metodologia hierárquica e, também, utilizaram o algoritmo *furthest-neighbor* (DUDA, 1973) (EVERITT, 1974) para computar o otimizado *cluster* (*dendogram*).

Em outros estudos, há autores (XIONG, 2009) que fizeram uma avaliação de desempenho de determinado serviço, utilizando QOS para garantir a qualidade de entrega de serviços pelo provedor do novo paradigma computacional. Com base nesse novo paradigma, os autores buscaram respostas para três questões:

1. Para um determinado recurso, em qual nível de serviço o QOS pode ser garantido?
2. Para um determinado número de clientes, quantos recursos de serviços serão necessários para assegurar que o serviço para o cliente será garantido em termos de tempo de resposta em percentuais?
3. Para um determinado recurso de serviço, quantos clientes podem ser suportados em termos de tempo de resposta em percentuais?

Outros autores (MAMIDALA, 2008) exploraram a utilização de algoritmos de

comunicação coletiva do MPI (`MPI_BCAST`, `MPI_AllGather`, `MPI_AllReduce` e `MPI_AllToAll`) sobre duas arquiteturas de processadores: Intel Clovertown e AMD Opteron. Os autores também caracterizaram o comportamento desses algoritmos coletivos em ambientes multiprocessadores com concorrência de rede e com comunicação *intra-node*. Essa otimização reduziu a latência do `MPI_BCAST` em 1,9 vezes o tempo de execução; já o `MPI_AllGather` foi de 4,0 vezes sobre 512 processadores. O `MPI_AllReduce` foi melhorado em 33%. Os autores também fizeram um teste usando o algoritmo de multiplicação de matrizes, coletando dados por meio de um *benchmark,* e obtiveram uma melhora de 3 vezes no desempenho.

Também há estudos (BOSC, 1995) que apresentaram uma solução sobre o modelo relacional para representar informações imprecisas. Para isso, foram sugeridas duas possibilidades: a primeira consiste em fazer consultas (*queries*) imprecisas. A segunda utiliza a teoria Fuzzy. Por meio desta lógica, o autor conceituou, para ambas as possibilidades, a Fuzzy aplicando a teoria para criar tipos de dados, categorizando uma série de dados, cada qual por meio de algum critério.

Ainda é possível observar (VAQUERO, 2011) os desafios de escalabilidade de aplicação em ambiente de nuvem. Há estudos que trouxeram abordagens de estruturas de `PaaS` e `IaaS` para explorar as dificuldades que cada uma dessas estruturas tem apresentado na atualidade. Escalabilidade é um dos conceitos que a computação nas nuvens propõe como solução, provocando a sensação de recursos infinitos. Porém, as formas de escalabilidade ainda são objetos de discussão e argumentação, pois não há consenso sobre o modo de escalar aplicações, servidores, redes, plataforma e bancos de dados. Os autores propõem uma forma de escalar no nível do `IaaS` por meio do *load balancing* (LB), que distribui a carga de execução das aplicações em diferentes servidores, escalando as MVs conforme a necessidade de processamento da LB. Já na abordagem no nível do `PaaS`, sugeriu-se utilizar replicação por contêiner ou por banco de dados – situação que pode ser explorada com mais profundidade, pois o `PaaS` apresenta diversos itens que precisam ser endereçados, em termos de segurança e desempenho (pelo fato de a replicação utilizar método de sincronização), demandando

banda de rede. Contudo, com o LB no nível do `IaaS`, esta é uma abordagem que possibilitou escalar os recursos de modo interessante, mas que ainda requer alguns estudos para se tornar uma solução de escalabilidade eficiente em ambiente de nuvem de infraestrutura.

Outras pesquisas (TICKOO, 2010) apresentam uma análise sobre os desafios na modelagem de desempenho de máquinas virtuais em um *data center*. Basicamente, nas estruturas de servidores multiprocessadores e na ferramenta de *benchmark* de VM vConsolidate, identificaram-se três problemas: modelagem da contenção dos dispositivos visíveis (CPU, capacidade de memória, dispositivos de I/O etc.), de recursos invisíveis (recursos de microarquitetura compartilhada, *cache* compartilhado, memória compartilhada etc.), e o *overhead* do VMM (ou *hypervisor*). Os autores chegam a conclusões relevantes, segundo as quais modelar as interferências, os recursos visíveis e invisíveis pode influenciar significativamente no desempenho da MV.

3.6 CONCLUSÃO DO CAPÍTULO

Este capítulo foi apresentado a proposta da metodologia de caracterização de aplicações por taxonomias ou características similares. Foi feito experimentos para comprovar o método e obtido dados sobre a caracterização de aplicações. Além de apresentar os experimentos instrumentados pelo método proposto. Foi apresentado trabalhos relacionados com as teorias e experimentos similares com esta dissertação. No próximo capítulo será apresentado os resultados e analises dos experimentos para comprovar o estudo desta metodologia proposta.

4. ANÁLISE E RESULTADOS EXPERIMENTAIS

Este capítulo apresenta as análises dos resultados coletados nos experimentos executados com base na metodologia de caracterização de aplicações em MVs. Com esse estudo, objetiva-se alcançar números que possam justificar a implementação ou adoção deste método para avaliação e previsão de utilização de recursos computacionais para ambientes de computação em nuvem. Os experimentos foram definidos e estruturados para examinar aplicações da comunidade que possam ser utilizadas para avaliação de desempenho. Uma vez definido, pode-se instrumentar, caracterizar e comprovar que o método condiz com o comportamento real desta aplicação em questão. As aplicações `pbzip2` e `lame` foram escolhidas para serem avaliadas em ambiente de MVs, sendo que o `pbzip2` pode executar a compactação utilizando *threads* para particionar o processo de compressão de dados. Já o `lame` faz a conversão de arquivos de música em processo serial, possibilitando avaliar o comportamento de um processo paralelo e serial. A seguir, os conceitos detalhados das duas aplicações do experimento.

4.1 APLICAÇÃO PBZIP2

O `pbzip2` (PBZIP, 2011) é uma implementação do `bzip2`, mas para processamento *multi-threading*, com o intuito de paralelizar a compactação de um determinado arquivo. O `pbzip2` utiliza recursos de *pthreads* para particionar os arquivos por meio de *block-sorting* e atribui a *thread* para ser executada com ou sem multiprocessadores. O `bzip2`, por sua vez, tem o mesmo algoritmo de compactação, porém emprega apenas um processador (*single core*). Ambas as aplicações estão no modelo de licença *Open Source* (`BSD-style`), com restrições para comercialização, e, por isso, é necessário questionar a comunidade BSD para fins comerciais. Por outro lado, para a comunidade acadêmica, pode ser utilizado para fins exploratórios, experiências de desenvolvimento e aprimoramento da ferramenta ou estudos em outros ambientes. Pelo fato de o `pbzip2` suportar múltiplos processos executados em paralelo, pode-se avaliar a intensidade média da utilização da CPU no processamento

da compactação de um arquivo com 4.3 Gb de dados (ISO do Opensuse V11.4). Além disso, esse produto tem uma elevada utilização de memória (para o processo de compactação) e grande volume de leitura e escrita em disco na geração do arquivo compactado.

A seguir, são evidenciados os parâmetros do comando `pbzip2`, fornecidos para a execução de compactação ou descompactação de arquivos.

Tabela 22: Opções do pbzip2

Opção	Descrição
`-b#`	# é o tamanho do bloco de 100 Kb por sequência (o padrão é 9, pois representa 900Kb).
`-c` ou `--stdout`	Envia o status do `pbzip2` para o `stdout` (terminal).
`-d` ou `--decompress`	Descompacta um aquivo `bzip`.
`-f` ou `--force`	Força a sobreposição de um arquivo existente.
`-h` ou `--help`	Imprime no `stdout` a mensagem de ajuda com os parâmetros da Tabela 22.
`-k` ou `--keep`	Mantém o arquivo de entrada e não remove o arquivo após a compactação.
`-l` ou `--loadavg`	Média de carga determinada pelo número máximo de processadores disponíveis para serem utilizados.
`-m#`	`#` é o máximo de memória a ser utilizado a cada Mb por sequência (o padrão é 100 que presenta 100 Mb de memória)
`-p#`	`#` é o número de processadores (o padrão é autodetectar).
`-q` ou `--quiet`	Modo silencioso (não apresenta informações detalhadas no `stdout`).
`-r` ou `--read`	Lê o arquivo de entrada inteiro e depois o divide entre os processadores alocados.
`-S#`	É o tamanho da *Thread* filha em 1Kb por sequência (padrão

	do tamanho da *thread* caso não seja especificado nenhum valor).
`-t` ou `--test`	Validação da integridade do arquivo compactado.
`-v` ou `--verbose`	Modo detalhado. Envia todos os detalhes da compactação para o `stdout`.
`-V` ou `--version`	Apresenta no `stdout` as informações sobre a versão do `pbzip2`.
`-z` ou `--compress`	Compacta o arquivo (o padrão do `pbzip2` é compactar caso não seja informado o parâmetro `-d` ou `-decompress`)
`-1` ou `--fast` … `-9` ou `--best`	Define o tamanho do bloco Algoritmo Burrows-Wheeler Transform (BWT) para 100 Kb … 900 Kb (o padrão é 900 Kb).
`--ignore-trailing-garbage=#`	`#` pode ter valor 1 (ignora) e 0 (proibido). Sendo que *trailing-garbage* são os textos ou valores extras utilizados na compactação ou descompactação dos arquivos.

A Tabela 23 apresenta um exemplo de compactação do arquivo ISO do `opensuse` 11.4 (4.6 Gb) por meio do `pbzip2` com os seguintes parâmetros:

```
pbzip2 -kzv -p4 -m1024.
```

Tabela 23: Exemplo de compactação do `pbzip2`

```
Parallel BZIP2 v1.1.3 - by: Jeff Gilchrist [http://compression.ca]
[Mar. 27, 2011]             (uses libbzip2 by Julian Seward)
Major contributions: Yavor Nikolov <nikolov.javor+pbzip2@gmail.com>
        # CPUs: 4
BWT Block Size: 900 KB
File Block Size: 900 KB
Maximum Memory: 1024 MB
-------------------------------------------
        File #: 1 of 1
     Input Name: /personal/nfsom/opensuse/openSUSE-11.4-DVD-x86_64.iso
```

```
    Output Name: /personal/nfsom/opensuse/openSUSE-11.4-DVD-x86_64.iso.bz2
     Input Size: 4614782976 bytes
Compressing data...
    Output Size: 4553793457 bytes
-----------------------------------------
     Wall Clock: 650.454242 seconds
Parallel BZIP2 v1.1.3 - by: Jeff Gilchrist [http://compression.ca]
[Mar. 27, 2011]             (uses libbzip2 by Julian Seward)
Major contributions: Yavor Nikolov <nikolov.javor+pbzip2@gmail.com>
```

A saída da Tabela 23 é apresentada assim que o processo de compactação é finalizado. Um dado a ser utilizado para efeitos comparativos é a medida *Wall Clock*, que indica o tempo em segundos que se levou para processar a compactação do arquivo.

4.2 APLICAÇÃO LAME

O `lame` (LAME, 2011) é uma ferramenta de conversão de arquivos de áudio (música ou vídeo). Este aplicativo possui características de *enconder* para tipos diferentes de arquivos de música ou vídeo (`mpeg`, `wav`), manipula a qualidade da música, retira os ruídos, muda a frequência, altera o `mp3` *tag*, entre outras modificações em arquivos de mídia. A utilização dos recursos computacionais neste aplicativo é limitada pelo fato da conversão não exigir processamento intenso ou recursos de memória e disco. Muitas vezes, os arquivos de música não possuem grandes quantidades de *bytes*, o que facilita o tratamento do arquivo diretamente em memória sem utilizar o disco para *swapping*. Porém, para arquivos com vídeo e música, a conversão demandará mais recursos computacionais, pois possuem mais dados a serem manipulados.

A seguir, a Tabela 24 mostra os parâmetros que o comando `lame` fornece para converter arquivos de áudio.

Tabela 24: Parâmetros do `lame`

Opção	Descrição
`-b #`	Define o a taxa de bits (bitrate), o padrão é 128 Kb.
`-h`	Melhor qualidade, porém torna o processamento mais lento.
`-f`	Modo rápido, porém de baixa qualidade.
`-V #`	Definição da qualidade por VBR (Variable Bit Rate). O Padrão é 4. 0 = maior qualidade, arquivos maiores. 9 = menor qualidade, arquivos menores.
`--preset type`	O tipo (type) pode ser *medium*, *standard*, *extreme*, *insane*, ou um valor médio desejado de taxa de bit (*bitrate*) e dependendo do valor especificado, um valor de qualidade será configurado e utilizado na conversão.
`--preset help`	Provê mais informação sobre o --preset.
`--longhelp`	Apresenta a mensagem de ajuda completa do lame.
`--license`	Apresenta informações sobre a licença.

A seguir, um exemplo de saída (*stdout*) do `lame`, sendo que foi convertido um arquivo wav para mp3 usando apenas a opção: `lame -h`.

Tabela 25: Exemplo de saída (`stdout`) do `lame`

```
LAME 3.98.4 64bits (http://www.mp3dev.org/)
Using polyphase lowpass filter, transition band: 16538 Hz - 17071 Hz
Encoding /personal/local/music/Adagio_molto_e_cantabile.wav
     to /personal/local/music/Adagio_molto_e_cantabile.wav.mp3
Encoding as 44.1 kHz j-stereo MPEG-1 Layer III (11x) 128 kbps qval=2
   Frame          |  CPU time/estim | REAL time/estim | play/CPU |    ETA
 39273/39273 (100%)|    1:41/    1:41|    1:41/    1:41|   10.117x|    0:00
------------------------------------------------------------------------------
   kbps       LR    MS  %     long switch short %
  128.0       13.3  86.7       100.0   0.0   0.0
Writing LAME Tag...done
ReplayGain: +7.3dB
```

Na Tabela 25, foi apresentado o resultado da conversão do arquivo, cujo tempo de conversão foi de 1 minuto e 41 segundos. A medida a ser utilizada é REAL time, pois apresenta o tempo real que o lame levou para processar a conversão do arquivo wav para mp3.

4.3 EXPERIMENTOS EXECUTADOS

As três fases do experimento foram avaliadas por meio de execuções dos programas `pbzip2` e `lame`. Em todas, foi preciso se certificar de que o *cache* do *kernel* foi reiniciado em cada execução das aplicações, para que ele não influencie no momento da implementação. Pelo fato de buscar uma análise do desempenho de aplicações, o *cache* pode ser um fator otimizador que esconde o desempenho real da aplicação. A seguir, é descrito o comando para reiniciar o *cache* no *kernel*.

```
echo 3 > /proc/sys/vm/drop_caches
```

Outro cuidado necessário para não influenciar na instrumentação foi a forma de executar as iterações dos dois programas. Primeiramente, foi executado o pbzip2, pois este leva um tempo elevado para processar a compactação do arquivo. Portanto, os cenários foram executados na seguinte ordem:

Tabela 26: Cenários de execução do `pbzip2`

Número de *threads*	Quantidade de memória
4	1024 Mb
3	1024 Mb
2	1024 Mb
1	1024 Mb
4	512 Mb
3	512 Mb
2	512 Mb
1	512 Mb

Na Tabela 26, cada cenário foi implementado por intermédio de um *script bash* que executou, na ordem da tabela, duas iterações em um laço de repetição (`for`), modificando os parâmetros `-p` e `-m` para variar o valor de *threads* e memória, respectivamente. Esse algoritmo foi executado quatro vezes em cada cenário, coletando oito amostras para análise. Pelo fato de os cenários não terem tido diferença significativa, não houve necessidade de coletar outras amostras. Com oito execuções de 20 iterações cada, o planejado foi o suficiente para obter dados do comportamento da aplicação `pbzip2` em MVs. Por fim, foram executados os cenários com a abordagem local (na própria MV), depois o cenário com o dom0 (NFS dom0) e, finalmente, NFS com uma máquina remota (NFS remoto).

Com relação ao programa de conversão l`ame`, este foi processado de modo diferente em relação ao experimento anterior; principalmente pelo fato de não possuir suporte a multiprocessador, a MV foi reconfigurada para *single core*. Além disso, o comando não possuía opção de modificação de memória máxima alocada para a aplicação, o que dificulta a utilização de um *script bash* para processar todos os cenários de uma vez. Portanto, conforme a Tabela 27, foram definidos os seguintes cenários para o `lame`:

Tabela 27: Cenários do experimento da aplicação `lame`

Cenário	Quantidade de memória
Local	512 Mb
	1024 Mb
NFS dom0	512 Mb
	1024 Mb
NFS remoto	512 Mb
	1024 Mb

Para a conversão dos arquivos de `wav` para mp3, foi escolhida a 9ª sinfonia do compositor Ludwig van Beethoven, com quatro músicas no álbum em formato wav. A primeira música é "*Allegro ma non troppo, un poco maestoso*" (175 Mb), a segunda é

“*Adagio molto e cantabile*” (173 Mb), a terceira é “*Molto vivace*” (145 Mb), e a última, “*Presto*” (242 Mb).

Primeiramente, foi executado um *script bash* que implementou dez iterações de cada cenário, alternando as execuções entre local, dom0 e remoto com a configuração de 1024 Mb de memória na MV. Pelo fato de a conversão dos arquivos ser rápida, é possível experimentar mais iterações para identificar desvios que possam caracterizar algum possível problema que precisaria de análise. Posteriormente, a MV foi desligada e a alocação de memória reconfigurada para 512 Mb. Sendo assim, o experimento foi reiniciado com a nova quantidade de memória. Ao se variar a memória, objetivou-se verificar se ela influencia na execução da aplicação.

4.4 RESULTADOS OBTIDOS

A seguir será apresentado os resultados obtidos nos experimentos nos cenários apresentados no capítulo 4.3.

4.4.1 Resultados do `pbzip2`

O experimento do `pbzip2` levou a situações que não apresentaram os resultados esperados. Portanto, eles precisam ser analisados por outros meios além do tempo de execução. O comando `pbzip2`, por suas características básicas, tem necessidade de processamento de CPU intenso, pelo fato desse comando utilizar algoritmos de compactação de arquivos. Sendo assim, esse comando, ao processar o algoritmo, solicita a alocação da CPU para o sistema operacional, efetuando assim a compactação e utilizando a memória e o HD conforme a necessidade e o tamanho dos arquivos a serem compactados.

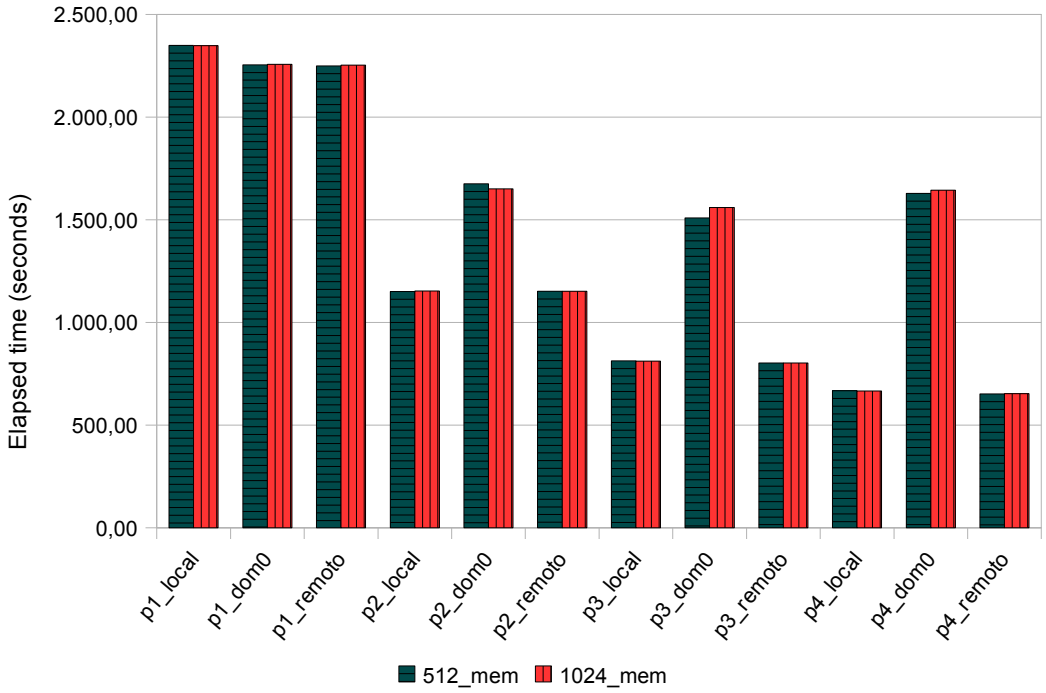

Figura 16: pbzip2 – tempo de execução

Na Figura 16, pode-se observar que não houve variação significativa no quesito tempo de execução, no que diz respeito aos cenários que variam a quantidade de memória alocada para a compactação do arquivo (ISO) de instalação do OpenSuse (4.3 Gb). Portanto, para essa situação, a quantidade de memória não influencia no resultado do `pbzip2`. Com relação ao FS utilizado, pode-se obter algumas conclusões importantes. Em todos os cenários de NFS com a máquina remota, obteve-se um tempo mais eficiente em relação aos outros FS. Esse resultado foi inesperado, uma vez que, nos experimentos anteriores, em todas as situações de execução de aplicações, de tipo multiprocessador ou distribuído, o NFS com uma máquina remota não foi indicado em nenhuma hipótese. Então, para esse experimento, faz-se necessária uma análise mais detalhada sobre o motivo desse desempenho comparado a outros cenários.

Para isso, foram utilizadas as saídas do comando de instrumentação `iostat` para analisar o *workload* que esse comando requisitou para processar a compactação.

O comando `iostat` fornece duas medidas que podem justificar o motivo do atraso para o processamento, tanto para a abordagem local quanto para o NFS com dom0: o I/O efetivo do FS e o I/O *wait*, que a aplicação necessitou efetuar, incluindo as interrupções ou gargalos no processamento no dom0 ou no próprio FS.

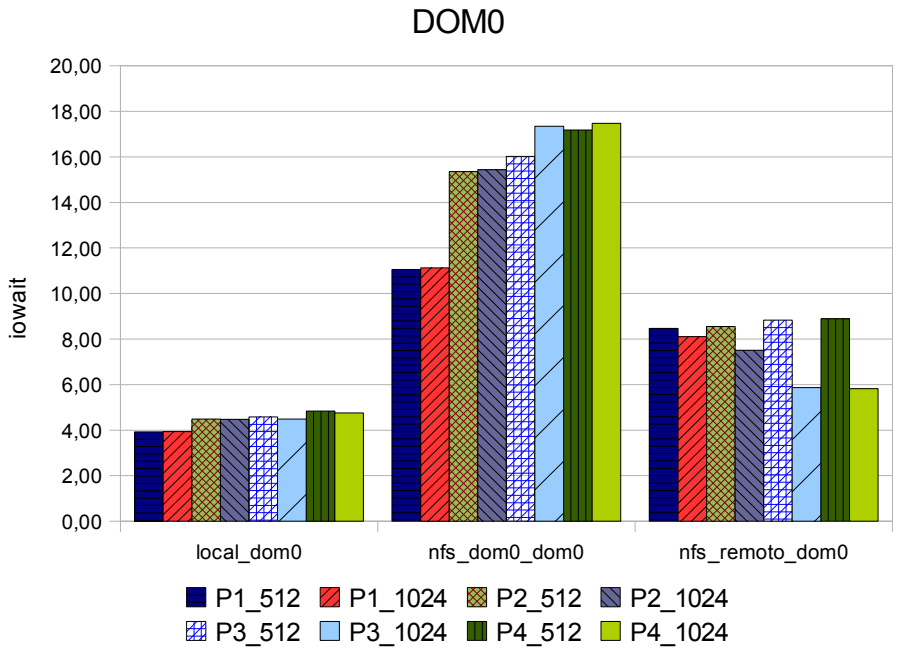

Figura 17: pbzip2 – tempo iowait do dom0

Conforme a Figura 17, o gráfico permite observar que o dom0 teve um processamento elevado para a abordagem do NFS com o dom0, fazendo todos os processos e a quantidade de memória terem um I/O *wait* elevado, por conta de um gargalo no dom0, influenciando em seu processamento. Uma vez que o dom0 é responsável pelo processamento de tudo que a MV solicita, ele fica impossibilitado de executar neste momento porque está com sobrecarga (*overhead*); consequentemente, a aplicação sofrerá atraso.

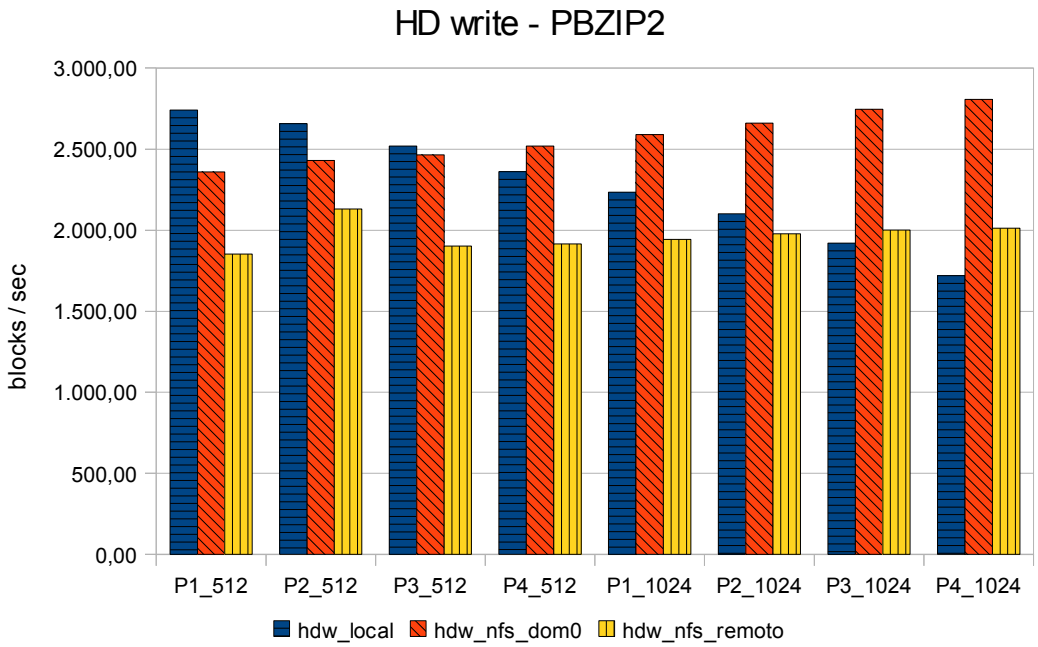

Figura 18: pbzip2 – I/O de gravação

Outra análise de I/O é apresentada na Figura 18, a medida de blocos por segundo gravados no FS. É possível observar que o I/O de gravação para o NFS com o dom0 permaneceu justificando que o motivo de atraso no processamento é o gargalo no dom0. Inclusive na quantidade de processos, há um aumento de I/O de gravação, pois o dom0 não conseguiu efetivar o I/O logo que foi requisitado. Outro dado interessante para analisar seria o I/O de gravação do NFS remoto. Foi observado que esteve na mesma faixa de I/O de gravação, não havendo muita variação entre os cenários. Uma possível justificativa para esse fenômeno é a distribuição do I/O de gravação com o servidor NFS remoto, o que não acontece quando o NFS está com o dom0, porque este é responsável por efetivar a gravação. Portanto, em se tratando da métrica de gravação para a situação de NFS com o dom0, isso não é recomendado.

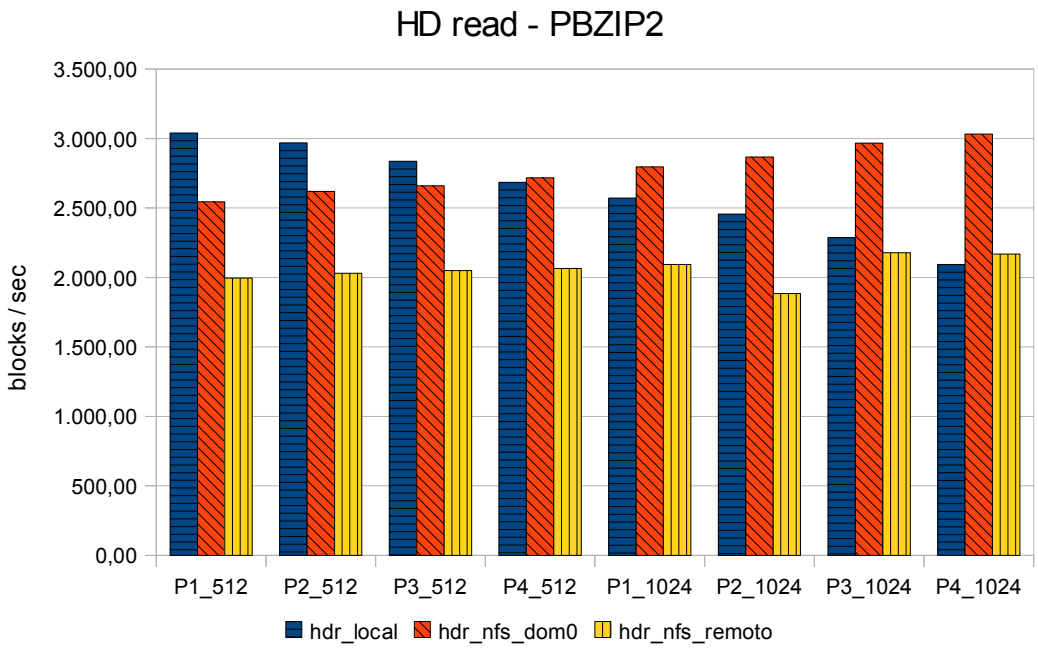

Figura 19: pbzip2 – I/O de leitura

Com relação à Figura 19, a métrica de leitura teve praticamente o mesmo comportamento da métrica de gravação. Portanto, mais uma justificativa para não adotar aplicações que requerem processamento elevado, usando a configuração de NFS com o dom0.

4.4.2 Resultados do lame

O experimento da aplicação `lame` obteve um comportamento esperado, sem nenhum desvio ou coleta inexplicável que demandasse uma análise detalhada sobre a execução da aplicação em todos os cenários executados.

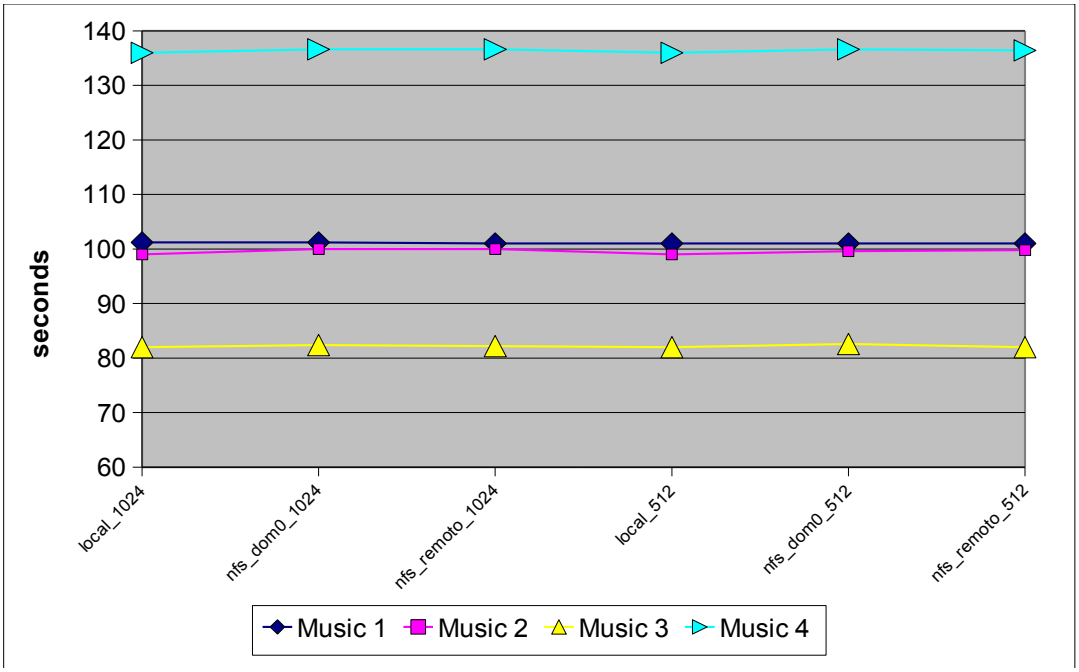

Figura 20: lame - tempo de execução

Segundo a Figura 20, é apresentado o gráfico com o tempo de execução da conversão dos arquivos wav para mp3 dos cenários da Tabela 27. Cada tempo foi proporcional ao tamanho dos arquivos (música 1 → 175 Mb; música 2 → 173 Mb; música 3 → 145 Mb; música 4 → 242 Mb), sendo que o comportamento de cada cenário obteve média aritmética muito próxima. Mesmo variando o FS em que o arquivo mp3 é gerado, assim como a quantidade de memória (512 Mb e 1024 Mb), não houve nenhuma situação em que o tempo de conversão esteve fora da normalidade. Portanto, para aplicações que não requerem muitos recursos de processamento, memória ou rede, não importa em qual tipo de FS o arquivo é gerado: é possível dispor qualquer solução de FS e CPU, e a aplicação terá o mesmo desempenho.

5. CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Esta dissertação objetivou apresentar uma metodologia de caracterização de aplicações em ambiente de computação nas nuvens. Esta metodologia apresentou um processo de automatização para avaliar e identificar a configuração ideal de MV. O provedor da nuvem ao aderir essa proposta de metodologia em sua infraestrutura, ele tem a possibilidade de controlar os processamentos das aplicações que estão sendo submetidos em sua infraestrutura. Para qualquer camada da nuvem (SaaS, PaaS e IaaS) haverá aplicações submetidas para processamento a todo momento pelos seus clientes da nuvem. Com esse controle, é feito uma administração de processamento na infraestrutura e cria um processo de elasticidade de recursos computacionais. Ou seja, assim que o cliente da nuvem necessitar de mais ou menos recursos computacionais, o provedor reconfigura a MV aumentando ou diminuindo o recurso computacional conforme a requisição. Na perspectiva do cliente da nuvem, é possível ter ganhos também. O cliente submete a aplicação para processamento em qualquer camada da nuvem (SaaS, PaaS e IaaS), o provedor identifica a configuração ideal seguindo a metodologia. E por fim, a aplicação é processada em uma MV configurada de acordo com a classificação que a metodologia propôs.

Esta dissertação foi dividida em duas fases, a primeira foi experimentada aplicações paralelas e distribuídas, e transacionais para obter dados de comportamento das aplicações em determinadas configurações de MVs. A segunda fase, foi definida a metodologia para caracterizar aplicações por meio de mecanismos de instrumentação e classificação. Pode-se aproveitar os resultados obtidos no experimento da primeira fase para definir a configuração para cada taxonomia. Para comprovar essa teoria, os dois experimentos com `pbzip2` e `lame` levam às seguintes conclusões: no caso do `lame`, que obteve baixa CPU, baixa memória, baixo I/O de disco, provou-se que não há influência na variação do FS ou da memória. O tempo de processamento foi similar de um cenário para outro. Sendo assim, para esse tipo de aplicação, pode-se utilizar o NFS com uma máquina remota, pois não haverá atraso na conversão do arquivo. Já com relação ao `pbzip2`, obtiveram-se outros resultados, sendo que o desempenho do NFS com o dom0 foi ruim, em virtude do gargalo de processamento e I/O do dom0, pois

este precisou processar o algoritmo de compactação, gerou I/O para ler o arquivo de origem e I/O para gravar efetivamente o arquivo no diretório NFS com o dom0. Portanto, o dom0 ficou sobrecarregado. Porém, essa mesma situação não ocorreu na abordagem de NFS com uma máquina remota, pelo fato de este ter distribuído o I/O de disco em equipamentos diferentes, portanto, o dom0 não teve gargalo significativo para atrasar o processamento. Uma MV alocada com abordagem distribuída com o mesmo dom0, a princípio, aparenta ser uma boa implementação, desde que a aplicação não exija processamento do dom0. Portanto, este método possibilita definir a melhor configuração de MVs de acordo com as características das aplicações, além de identificar gargalos de processamento ou consumo elevado de recurso computacional. Uma vez coletadas as informações da instrumentação, pode-se fazer uma abordagem analítica e encontrar alternativas para implementar um ambiente adequado para a aplicação, alterando a saída de configuração recomendada para o grupo em que a aplicação analisada se encontra. Com isso, o método torna-se flexível e incremental, possibilitando que o provedor da nuvem controle a alocação dos equipamentos de uma forma organizada e controlada.

5.1 CONTRIBUIÇÕES

A seguir, são expostas as contribuições alcançadas nesta dissertação:

I. Definida a estratégia das medições dos *benchmarks* e métricas para avaliação dos desempenhos das aplicações.
II. OpenNebula no OpenSuse 64 bits implementado e configurado. Soluções de *bugs* e melhorias na documentação do OpenNebula foram desenvolvidas e enviadas para a comunidade OpenNebula. Além disso, foi gerado um guia de instalação especificamente para OpenSuse, porque este requer alguns pacotes diferenciados com pré-requisitos para o funcionamento do OpenNebula.
III. Os cenários propostos foram executados e analisados, e a configuração ideal foi apresentada para as categorias de aplicações – exceto a categoria de rede, que ficou como trabalho futuro.

IV. Definido o método de instrumentação das aplicações, efetuando as leituras e coletas das medidas em paralelo com a execução do programa.
V. Definido o método de classificação (*fuzzy*) de acordo com o histórico das outras execuções.
VI. Foi feita uma análise dos programas `lame` e `pbzip2` por meio desse método. Foram encontradas situações caracterizando diferentes abordagens, e identificando particularidades.

5.2 PUBLICAÇÕES

Neste Capitulo é apresentado as apresentações e publicações de artigos alcançados durante o curso de pós-graduação:

- Uma primeira versão dos resultados preliminares foi publicado e apresentado em formato de artigo (OGURA, 2010a) no ERAD (Escola Regional de Alto Desempenho de São Paulo) SP 2010.
- Uma segunda versão dos resultados preliminares foi produzido artigo e publicado em formato de artigo (OGURA, 2010b) no simpósio Scientific and Engineering Computing (SEC), um subevento do CSE 2010 em conjunto com o IEEE; o dito artigo foi apresentado entre os dias 11 e 13 de dezembro de 2010.
- Outro artigo foi produzido com os dados obtidos da metodologia proposta na segunda etapa desta dissertação. Este artigo foi publicado e apresentado no ERAD (Escola Regional de Alto Desempenho de São Paulo) SP 2011.

5.3 TRABALHOS FUTUROS

Esta dissertação apresentou uma metodologia de caracterização de aplicações que pode ser estendida e aplicada para outros ambientes além da nuvem. Este método também permite fazer uma avaliação inicial de qualquer aplicação, identificando certas características de I/O ou qualquer outra métrica, o que possibilita uma avaliação de

qualidade dos programas antes da efetivação no ambiente de produção. Outra possibilidade seria testar o *workload* de um ambiente computacional e verificar se este possui as características que o experimento avaliou.

Como trabalho futuro, esta dissertação pode ser incrementada com outras métricas, além de CPU, memória ou I/O. Pode-se, por exemplo, avaliar o comportamento da rede em relação ao processo que está sendo implementado; além de outras medidas de rede, como *Round Trip Time* (RTT), *throughput* (vazão de dados por segundos), QoS, entre outras métricas.

Outra proposta seria aplicar o mesmo método utilizando outro monitor de máquinas virtuais (MMV), no sentido de verificar se há diferenças entre o Xen e outros monitores, como VMWare, Virtualbox, KVM etc. Adicionalmente, pode-se aplicar em ambientes com maior capacidade de processamento e recursos computacionais, variando a quantidade de CPU, memória e HD em maior escala, além de verificar a possibilidade de fazer avaliações com recursos de *high performance computing* (HPC), *storage*, *high availability* (HA) ou *cluster*.

Outra situação a ser avaliada é a das MVs em uma nuvem pública, pois esse experimento foi executado com o OpenNebula (nuvem privada). Pode-se também avaliar se, a partir desse mesmo cenário, é possível obter diferentes desempenhos quando é processado no IaaS de um provedor da nuvem.

Por fim, executar diferentes tipos de sistemas de arquivos, como, por exemplo: OCFS2, LUSTRE, GPFS, REISERFS, XFS, HADOOP, entre outros.

REFERÊNCIAS BIBLIOGRÁFICAS

AGARWAL, B.; Tayal, P.; Gupta, M.; **Software Engineering & testing**, Jones and Bartlett Publishers, 2008 , p. 110 – 112.

ARMBRUST, M. et al. **Above the Clouds: A Berkeley View of Cloud computing, Technical Report No. UCB/EECS-2009-28**. University of California at Berkley, 2009.

BAILEY, D.; BARSZCZ, E. **The NAS Parallel Benchmarks**. NASA TM 103863, Moffett Field. California: NASA Ames Research Center, 1993.

BARHAM, P.; et. al. **Xen and the Art of Virtualization.** Proceedings of the 19th ACM SOSP, 2003, p. 164-177.

BARUCHI, A.; MIDORIKAWA, E. **Memory Dispatcher: Uma Contribuição para a Gerência de Recursos em Ambientes Virtualizados**. 2010. Dissertação (Mestrado) - Escola Politécnica, Universidade de São Paulo, São Paulo, 2010.

BOSC. P.; PIVERT O. **SQLf: A relational database language for fuzzy querying**. IEEE Trans. on Fuzzy Systems. v. 3, n. 1, p. 1-17, 1995.

CHERKASOVA, L.; GARDNER, R. **Measuring CPU overhead for I/O processing in the Xen virtual machine monitor**, Proceedings of USENIX Annual Technical Conf, Apr 2005.

CHERKASOVA, L.; GUPTA, D.; Vahdat, **A. Comparison of the Three CPU Schedulers in Xen**, SIGMETRICS Perf. Eval. Rev. v. 35, n. 2, p.42-51, 2007.

DUDA, R.; HART, P. **Pattern Classification and Scene Analysis**, J. Wiley, 1973.

EL-GHAZAWI, T.; CANTONNET, F. **UPC performance and potential: A NPB experimental study**, Supercomputing2002 (SC2002), 2002.

EVERITT, B. **Cluster Analysis**. New York: Wiley ,1974.

FOSTER, I.; ZHAO, Y.; RAICU I.; LU, S. **Cloud computing and grid computing 360-**

degree compared. Grid Computing Environments Workshop, 2008. p.1-10.

GRABNER, R.; MIETKE, F.; REHM, W. **An MPICH2 Channel Device Implementation over VAPI on InfiniBand**, Proceedings of the International Parallel and Distributed Processing Symposium, 2004.

GROPP, W. Mpich2: **A new start for mpi implementations**, Proceedings of the 9th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface. London, 2002, p.7.

GUPTA, D.; CHERKASOVA, L.; GARDNER, R.; VAHDAT, A. **Enforcing Performance Isolation Across Virtual Machines in Xen.** Proceendings of the ACM/IFIP/USENIX 7th Intl. Middleware Conf., Melbourne, 2006.

HAIZEA. Disponível em: **haizea.cs.uchicago.edu/**. Acesso em: julho de 2010.

IRFAN, H. **Virtualization with KVM.** Linux J. 2008, p.166 .

JAIN, R. **The art of computer system performance analysis**. John Wiley & Sons, New York, 1991. p.3–44.

LAME. Disponível em: **lame.cvs.sourceforge.net/viewvc/lame/lame/USAGE**. Acesso em: abril de 2011.

LAUREANO, Marcos. **Máquinas Virtuais e Emuladores: Conceitos, Técnicas e Aplicações.** 1a Edição, São Paulo, Ed. Novatec. Junho, 2006.

LEE, C. **Fuzzy logic in control systems: Fuzzy logic controller (part i)**. IEEE Transactions on Systems, Man an Cybernetics. v.20, n.2, p.404-418, mar./abr. 1990.

MAMIDALA, A.; KUMAR, R.; DE, D.; PANDA, D. **Mpi collectives on modern multicore clusters: Performance optimizations and communication characteristics**, Int'l Symposium on Cluster Computing and the Grid. Lyon, 2008.

OGURA, D.; MIDORIKAWA, E. **Caracterização de aplicações científicas e**

transacionais em ambientes Cloud Computing. In: ERAD SP, 2010a.

OGURA, D.; MIDORIKAWA, E. **Characterization of Scientific and Transactional Applications under Multi-core Architectures on Cloud Computing Environment**. In: 13th IEEE International Conference on Computational Science and Engineering Workshop SEC 2010, Anais ISBN: 978-0-7695-4323-9 China, 2010b. p.314-320.

OGURA, D.; MIDORIKAWA, E. **Instrumentação de aplicação paralela em ambientes de computação na nuvem.** In: ERAD SP, 2011.

ORDUNA, J.; ARNAU, V.; DUATO, J. **Characterization of Communication between Processes in Message-passing Applications**, Proceedings of International Conference on Cluster Computing (Cluster-2000), 2000.

ORLANDO, S.; RUSSO, S. **Java Virtual Machine monitoring for dependability benchmarking**, *Ninth IEEE International Symposium,* pp., 24-26, April 2006

PARKHILL, D. **The challenge of the computer utility**, Addison-Wesley, Reading, 1966.

PATEL, P.; RANABAHU, A.; SHETH, A. **Service Level Agreement in Cloud Computing, Workshop at Object-Oriented Programming**, Systems, Languages & Applications, 2009.

PBZIP2. COMPRESSION CA. Disponível em: **www.compression.ca/pbzip2/**. Acesso em: abril de 2011.

SNIR, M.; OTTO, S.; HUSS-LEDERMAN, S.; WALKER, D.; DONGARRA, J. **MPI: The Complete Reference**. MIT Press, 1995.

SOTOMAYOR, B. et. al. **Capacity Leasing in Cloud Systems using the OpenNebula Engine.** Workshop on Cloud Computing and its Applications (CCA08).

TICKOO, O.; IYER, R.; ILLIKKAL, R.; NEWELL, D. **Modeling Virtual Machine Performance: Challenges and Approaches**, ACM SIGMETRICS, 2010.

TPC. TPC-H. Disponível em: **www.tpc.org/tpch/**. Acesso em: outubro de 2010.

VAQUERO, L. M.; RODERO-MERINO, L.; BUYYA, R. **Dynamically Scaling Applications in the Cloud**. ACM SIGCOMM Computer Communication Review, v.41, n.1, janeiro de 2011.

VMWARE ESX SERVER. Disponível em: **www.vmware.com/products/esx.** Acesso em: outubro de 2010.

VMWARE. **VMware Technical White Paper**. Palo Alto, CA – USA, 1999.

XIONG, K. et al, **Service Performance and Analysis in Cloud Computing**, IEEE. 6-10, p.693 – 700, julho de 2009

ZHANG, L.; CHENG, L.; BOUTABA, R. **Cloud computing: state-of-the-art and research challenges**, Journal of Internet Services and Applications, Springer. v. 1, n. 1, p. 7-18, 2010.

APÊNDICE A – *script* de criação do banco de dados

```
db2 CREATE DATABASE APPL_DB ON /images/db

db2 connect to APPL_DB

db2 "CREATE TABLE Instrumentacao (idInstr INTEGER NOT NULL ,
vlrMetrica decimal(10,2) NOT NULL, codMetrica INT NOT NULL,
codExecucao INT NOT NULL, primary key (idInstr, codMetrica,
codExecucao))"

db2 "CREATE TABLE Metrica (CodMetrica INTEGER NOT NULL
generated always as identity (start with 0, increment by 1, no
cache) primary key, dscMetrica VARCHAR(20) NOT NULL,
dscTaxonomia VARCHAR(4) NOT NULL)"

db2 "CREATE TABLE Execucao (CodExecucao INTEGER NOT NULL
generated always as identity (start with 0, increment by 1, no
cache) primary key, dscExecucao VARCHAR(50) NOT NULL,
codMaquina INTEGER NOT NULL, DataExecucao VARCHAR(20))"

db2 "CREATE TABLE Maquina (CodMaquina INTEGER NOT NULL
generated always as identity (start with 0, increment by 1, no
cache) primary key, Hostname VARCHAR(30) NOT NULL, IP
VARCHAR(15) NOT NULL, Memoria INTEGER NOT NULL, HD INTEGER NOT
NULL, Rede VARCHAR(30) NOT NULL, KERNEL VARCHAR(30) NOT NULL)"
```

APÊNDICE B – *script* de importação no banco de dados de instrumentação

```
#!/bin/bash
cd $1
dirname=${PWD##*/}

###### Importing iostat ######
output=`cat iostat_exp.out |egrep -A 1 "avg-cpu:" | awk '{if
(NR!=1 && $1!="avg-cpu:"&& $1!="--") {print $1}}'`

count=1
timestamp=`grep  "Estatisticas"  iostat_exp.out  |awk  '{print
$1}'`

#connecting to appl_db db2 database
db2 connect to appl_db

db2   "insert   into   execucao   (dscExecucao,   CodMaquina,
DataExecucao) VALUES ('$dirname', 0, '$timestamp')"

codexec=`db2 -x "select max(codexecucao)  from execucao where
dscExecucao = '$dirname'"`

for mem in `echo $output`
do
     db2 "insert into instrumentacao values ($count, $mem, 0,
'$codexec')"
     count=$((count+1))
done
```

```
##### Importing pmap #####
output=`cat pmap.out |egrep "writable-private" |awk '{print $1
" " $3}' | sed 's/ *K/ /g'`
count=1
wrprivate=1
for number in `echo $output`
do
  if [ $wrprivate = 1 ]; then
     wp=$number
     wrprivate=$((wrprivate+1))
  else
     rp=$number
         db2 "insert into instrumentacao values ($count, $
((wp+rp)), 1, $codexec)"
     wrprivate=1
     count=$((count+1))
  fi;
done

##### Importing iostat #####
output=`cat iostat_blk.out |egrep "xvda4" |awk '{print $3 " "
$4}'`
count=1
wrprivate=1

for number in `echo $output`
do
  if [ $wrprivate = 1 ]; then
      db2 "insert into instrumentacao values ($count, $number,
2, $codexec)"
```

```
     wrprivate=$((wrprivate+1))
  else
      db2 "insert into instrumentacao values ($count, $number,
3, $codexec)"
     wrprivate=1
     count=$((count+1))
  fi;
done
```

APÊNDICE C – *stored procedure* e *user function* para a classificação *Fuzzy*.

```
CREATE PROCEDURE P_CATEG_FUZZY (OUT cat VARCHAR(4), IN
vlrMetrica DECIMAL(10,2), IN codMetr INT)
LANGUAGE SQL
READS SQL DATA
BEGIN
  DECLARE v_qtde_reg INT DEFAULT 0;
  DECLARE v_qtde_reg_min INT DEFAULT 0;
  DECLARE v_qtde_reg_max INT DEFAULT 0;
  DECLARE v_qtde_real INT DEFAULT 0;
  DECLARE a_maior_max DECIMAL(10,2) DEFAULT 0;
  DECLARE a_medio_max DECIMAL(10,2) DEFAULT 0;
  DECLARE a_menor_max DECIMAL(10,2) DEFAULT 0;
  DECLARE dscMetr VARCHAR(4);
  SELECT COUNT(CodExecucao)/3 INTO v_qtde_reg FROM Execucao;
  SELECT COUNT(CodExecucao) INTO v_qtde_real FROM Execucao;
  IF ( v_qtde_real <= 3 ) THEN
     SET v_qtde_reg_max = v_qtde_real;
     SET v_qtde_reg_min = v_qtde_real;
  ELSE
      SET v_qtde_reg_max = v_qtde_real;
      SET v_qtde_reg_min = v_qtde_reg_max - v_qtde_reg;
  END IF;
   SET (a_maior_max) = (SELECT MIN(NUM_LINHA.avgVlrMetrica)
FROM Instrumentacao R INNER JOIN (SELECT ROW_NUMBER()
OVER(ORDER BY AVG(vlrMetrica) ASC) AS IDNUM, CodExecucao,
DECIMAL(AVG(vlrMetrica),10,2) AS avgVlrMetrica FROM
Instrumentacao WHERE CodMetrica = codMetr GROUP BY
```

```
CodExecucao) AS NUM_LINHA ON NUM_LINHA.CodExecucao =
R.Codexecucao AND NUM_LINHA.IDNUM BETWEEN  v_qtde_reg_min AND
v_qtde_reg_max);

  IF ( v_qtde_real > 1 AND v_qtde_real <=3 ) THEN
     SET v_qtde_reg_max = v_qtde_real - 1;
     SET v_qtde_reg_min = v_qtde_real - 1;
  ELSE
      SET v_qtde_reg_max = v_qtde_reg_min - 1;
      SET v_qtde_reg_min = v_qtde_reg_max - v_qtde_reg;
  END IF;

   SET (a_medio_max) = (SELECT MIN(NUM_LINHA.avgVlrMetrica)
FROM Instrumentacao R INNER JOIN (SELECT ROW_NUMBER()
OVER(ORDER BY AVG(vlrMetrica) ASC) AS IDNUM, CodExecucao,
DECIMAL(AVG(vlrMetrica),10,2) AS avgVlrMetrica FROM
Instrumentacao WHERE CodMetrica = codMetr GROUP BY
CodExecucao) AS NUM_LINHA ON NUM_LINHA.CodExecucao =
R.Codexecucao AND NUM_LINHA.IDNUM BETWEEN  v_qtde_reg_min AND
v_qtde_reg_max);

  IF ( v_qtde_real =  3 ) THEN
     SET v_qtde_reg_max = 1;
     SET v_qtde_reg_min = 1;
  ELSE
      SET v_qtde_reg_max = v_qtde_reg_min - 1;
      SET v_qtde_reg_min = 1;
  END IF;
   SET (a_menor_max) = (SELECT MIN(NUM_LINHA.avgVlrMetrica)
FROM Instrumentacao R INNER JOIN (SELECT ROW_NUMBER()
```

```
OVER(ORDER BY AVG(vlrMetrica) ASC) AS IDNUM, CodExecucao,
DECIMAL(AVG(vlrMetrica),10,2) AS avgVlrMetrica FROM
Instrumentacao WHERE CodMetrica = codMetr GROUP BY
CodExecucao) AS NUM_LINHA ON NUM_LINHA.CodExecucao =
R.Codexecucao AND NUM_LINHA.IDNUM BETWEEN  v_qtde_reg_min AND
v_qtde_reg_max);

   SET (dscMetr) = (SELECT dscTaxonomia FROM Metrica where
codMetrica = codMetr);
IF ( v_qtde_real = 1 ) THEN
    IF ( vlrMetrica >= a_maior_max ) THEN
        SET cat=dscMetr || '+';
    ELSE
         SET cat=dscMetr || '-';
    END IF;
ELSE
   IF ( v_qtde_real >= 2 ) THEN
       IF ( vlrMetrica >= a_maior_max ) THEN
           SET cat=dscMetr || '+';
        ELSE
             IF ( vlrMetrica >= a_medio_max AND vlrMetrica <=
a_maior_max ) THEN
                SET cat = dscMetr;
            ELSE
                SET cat=dscMetr || '-';
      END IF;
       END IF;
    END IF;
END IF;
END @
```

```
CREATE FUNCTION F_CATEG_FUZZY(vlrMetrica DECIMAL(10,2),
codMetr INT)
     RETURNS VARCHAR(4)
LANGUAGE SQL
READS SQL DATA
BEGIN ATOMIC
  DECLARE cat VARCHAR(4);
  CALL P_CATEG_FUZZY(cat , vlrMetrica, codMetr);
  RETURN cat;
END@
```

APÊNDICE D – *view* que apresenta a taxonomia para um determinado valor de I/O.

```
Db2 "CREATE VIEW vw_metrica AS
select decimal(avg(vlrmetrica),10,2) as avgmetrica,
decimal(stddev(vlrmetrica),10,2)  as  desviopad,m.dscmetrica,
i.codexecucao,F_CATEG_FUZZY(decimal(avg(vlrmetrica),10,2),
m.codmetrica) AS GRUPO from instrumentacao
i inner join metrica m ON i.codmetrica = m.codmetrica
where idInstr > 1 group by m.dscmetrica,
i.codexecucao, m.codmetrica"
```

APÊNDICE E – *script* de instrumentação – abordagem local

```
#!/bin/bash

###stat_scripts.sh###
NETSTAT="/personal/local/scripts/results/netstat_exp.out"
iostat="/personal/local/scripts/results/iostat_exp.out"
VMSTAT="/personal/local/scripts/results/vmstat_exp.out"
pmap="/personal/local/scripts/results/pmap.out"
iostat_BLK="/personal/local/scripts/results/iostat_blk.out"

> $NETSTAT
> $iostat
> $VMSTAT
> $pmap
> $iostat_BLK

#$1 is the process name
#$2 is the device name
#$3 is the iteration number
#$4 is the data time stamp for

echo $4 " Estatisticas de NETWORK "> $NETSTAT
echo $4 " Estatisticas de IO $2 "> $iostat
echo $4 " Estatisticas de MEM Virtual "> $VMSTAT
echo $4 " Estatisticas de Memoria " > $pmap
echo $4 " Estatisticas de IO BLOCKS $2 " > $iostat_BLK

# execute the xentop and iostat on dom0
ssh 192.168.1.102 "xentop -b >> /root/xentop_local$4_i$3.out"
&
ssh 192.168.1.102 "iostat /dev/sda4 -d 2 >>
/root/iostat_local$4_i$3_blk.out" &
ssh 192.168.1.102 "iostat /dev/sda4 -x -d 2 >>
/root/iostat_local$4_i$3_exp.out" &

while true; do
  pid=`pgrep $1`
  netstat -i >> $NETSTAT
  iostat $2 -x 1 1  >> $iostat
  iostat $2 1 1 >>$iostat_BLK
  vmstat 1 1 >> $VMSTAT

  if [ "$pid" != "" ]; then
```

```
      pmap $pid >> $pmap
  fi;
  sleep 1
done
```

APÊNDICE F - *script* de instrumentação - abordagem NFS com o DOM0

```
#!/bin/bash

###stat_scripts.sh###
NETSTAT="/personal/nfs/scripts/results/netstat_exp.out"
iostat="/personal/nfs/scripts/results/iostat_exp.out"
VMSTAT="/personal/nfs/scripts/results/vmstat_exp.out"
pmap="/personal/nfs/scripts/results/pmap.out"
iostat_BLK="/personal/nfs/scripts/results/iostat_blk.out"

> $NETSTAT
> $iostat
> $VMSTAT
> $pmap
> $iostat_BLK

echo $4 " Estatisticas de NETWORK "> $NETSTAT
echo $4 " Estatisticas de IO $2 "> $iostat
echo $4 " Estatisticas de MEM Virtual "> $VMSTAT
echo $4 " Estatisticas de Memoria " > $pmap
echo $4 " Estatisticas de IO BLOCKS $2 " > $iostat_BLK

# execute the xentop and iostat on dom0
ssh 192.168.1.102 "xentop -b >>
/root/xentop_nfs_dom0_$4_i$3.out" &
ssh 192.168.1.102 "iostat /dev/sda4 -d 2 >>
/root/iostat_nfs_dom0_$4_i$3_blk.out" &
ssh 192.168.1.102 "iostat /dev/sda4 -x -d 2 >>
/root/iostat_nfs_dom0_$4_i$3_exp.out" &

while true; do
  pid=`pgrep $1`
  netstat -i >> $NETSTAT
  iostat $2 -x 1 1  >> $iostat
  iostat $2 1 1 >>$iostat_BLK
  vmstat 1 1 >> $VMSTAT

  if [ "$pid" != "" ]; then
     pmap $pid >> $pmap
  fi;
  sleep 1
done
```

APÊNDICE G - *script* de instrumentação - abordagem NFS remoto

```
#!/bin/bash

###stat_scripts.sh###
NETSTAT="/personal/nfsom/scripts/results/netstat_exp.out"
iostat="/personal/nfsom/scripts/results/iostat_exp.out"
VMSTAT="/personal/nfsom/scripts/results/vmstat_exp.out"
pmap="/personal/nfsom/scripts/results/pmap.out"
iostat_BLK="/personal/nfsom/scripts/results/iostat_blk.out"

> $NETSTAT
> $iostat
> $VMSTAT
> $pmap
> $iostat_BLK

echo $4 " Estatisticas de NETWORK "> $NETSTAT
echo $4 " Estatisticas de IO $2 "> $iostat
echo $4 " Estatisticas de MEM Virtual "> $VMSTAT
echo $4 " Estatisticas de Memoria " > $pmap
echo $4 " Estatisticas de IO BLOCKS $2 " > $iostat_BLK

# execute the xentop and iostat on dom0
ssh 192.168.1.102 "xentop -b >>
/root/xentop_nfs_outra_maq_$4_i$3.out" &
ssh 192.168.1.102 "iostat /dev/sda4 -d 2 >>
/root/iostat_nfs_outra_maq_dom0_$4_i$3_blk.out" &
ssh 192.168.1.102 "iostat /dev/sda4 -x -d 2 >>
/root/iostat_nfs_outra_maq_dom0_$4_i$3_exp.out" &

#execute the iostat on nfs remote node. This is for getting
the IO of the NFS remote.
ssh 192.168.1.101 "iostat /dev/sdb -d 2 >>
/root/iostat_nfs_outra_maq_$4_i$3_blk.out" &
ssh 192.168.1.101 "iostat /dev/sdb -x -d 2 >>
/root/iostat_nfs_outra_maq_$4_i$3_exp.out" &

while true; do
  pid=`pgrep $1`
  netstat -i >> $NETSTAT
  iostat $2 -x 1 1  >> $iostat
  iostat $2 1 1 >>$iostat_BLK
  vmstat 1 1 >> $VMSTAT
```

```
  if [ "$pid" != "" ]; then
     pmap $pid >> $pmap
  fi;
  sleep 1
done
```

APÊNDICE H - *script* de execução do pbzip2

```
#!/bin/bash

# $dir variable is to indicate where is the location of the
opensuse ISO and scripts.
# LOCAL = /personal/local/opensuse
# NFS_DOM0 = /personal/nfs/opensuse
# NFS_REMOTE=/personal/nfsom/opensuse
dir='/personal/nfsom/opensuse'

# $hmdir variable is to indicate where is the location of the
pbzip/instrumentation scripts.
# LOCAL = /personal/local/scripts
# NFS_DOM0 = /personal/nfs/scripts
# NFS_REMOTE = /personal/nfsom/opensuse
hmdir='/personal/nfsom/scripts/'
 dt=`date +%y%m%d-%H%M%S`

 # clean up the kernel cache
 echo 3 > /proc/sys/vm/drop_caches

 #call the instrumentation script
 ./stat_scripts.sh pbzip2 /dev/xvda4 $3 $dt &
 #call the pbzip2 command
 pbzip2 -kzv -p$1 -m$2 $dir/openSUSE-11.4-DVD-x86_64.iso >
$hmdir/results/pbzip2.out

 #kill all instrumentation process
 pkill iostat
 pkill vmstat
 pkill netstat
 pkill pmap
 pkill stat_script
 ssh 192.168.1.102 "pkill xentop"
 ssh 192.168.1.101 "pkill iostat"
 ssh 192.168.1.102 "pkill iostat"

 mv $hmdir/results
$hmdir/results_pbzip_nfs_remoto_p$1_$2mem_$dt
 mkdir $hmdir/results
 rm  $dir/openSUSE-11.4-DVD-x86_64.iso.bz2
```

APÊNDICE I - script de execução do lame

```
#!/bin/bash
# $dir variable is to indicate where is the location of the
opensuse ISO and scripts.
# LOCAL = /personal/local/opensuse
# NFS_DOM0 = /personal/nfs/opensuse
# NFS_REMOTE=/personal/nfsom/opensuse
dir='/personal/nfsom/music'

# $hmdir variable is to indicate where is the location of the
pbzip/instrumentation scripts.
# LOCAL = /personal/local/scripts
# NFS_DOM0 = /personal/nfs/scripts
# NFS_REMOTE = /personal/nfsom/opensuse
hmdir='/personal/nfsom/scripts'

dt=`date +%y%m%d-%H%M%S`

# clean up the kernel cache
echo 3 > /proc/sys/vm/drop_caches
#call the instrumentation script
./stat_scripts.sh lame /dev/xvda4 $1 $dt &
# call the lame command to convert wav to mp3 all files
# in $dir
for file in $dir/*.wav; do
  lame -h $file $file'.mp3'
done

  #kill all instrumentation process
  pkill iostat
  pkill vmstat
  pkill netstat
  pkill pmap
  pkill stat_script
  ssh 192.168.1.102 "pkill xentop"
  ssh 192.168.1.101 "pkill iostat"
  ssh 192.168.1.102 "pkill iostat"

  mv $hmdir/results
$hmdir/lame_results_nfs_outra_maquina_1024mem_$dt
  mkdir $hmdir/results
```

APÊNDICE J - script de execução do MPI com NAS

```
#!/bin/bash

dt=`date +%y%m%d-%H%M%S`

for ((  i = 1 ;  i <= 20;  i++  )); do
 mpiexec -np $1 ./NPB3.3/NPB3.3-MPI/bin/is.C.$1 ./m_matriz >
results/results_$1_$i_$dt < /dev/null
done;
```

APÊNDICE K - *script* de execução do TPC-H

```
#!/bin/bash
db2 connect to TPCH2

for (( c=1; c<=20; c++ )); do
db2batch -d TPCD -f complexa.sql -r
results/res.txt,results/results_complexa_hd_$c.txt
rm results/res.txt

db2batch -d TPCD -f table_scan.sql -r
results/res.txt,results/results_table_scan_hd_$c.txt

rm results/res.txt

done
```