UNIVERSIDADE FEDERAL DO CEARÁ
DEPARTAMENTO DE COMPUTAÇÃO
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

PAULO ANTONIO LEAL REGO

FAIRCPU: UMA ARQUITETURA PARA PROVISIONAMENTO DE MÁQUINAS VIRTUAIS UTILIZANDO CARACTERÍSTICAS DE PROCESSAMENTO

FORTALEZA, CEARÁ

Março / 2012

PAULO ANTONIO LEAL REGO

FAIRCPU: UMA ARQUITETURA PARA PROVISIONAMENTO DE MÁQUINAS VIRTUAIS UTILIZANDO CARACTERÍSTICAS DE PROCESSAMENTO

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal do Ceará, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Prof. Dr. José Neuman de Souza

FORTALEZA, CEARÁ

Março / 2012

A000z REGO, P. A. L..

FairCPU: uma arquitetura para provisionamento de máquinas virtuais utilizando características de processamento / Paulo Antonio Leal Rego. Março / 2012.

81p.;il. color. enc.

Orientador: Prof. Dr. José Neuman de Souza

Co-Orientador:

Dissertação(Pós-Graduação em Ciência da Computação) - Universidade Federal do Ceará, Departamento de Computação, Fortaleza, Março / 2012.

1. Provisionamento de máquinas virtuais 2. Ambiente heterogêneo 3. Limitação do uso da CPU I. Prof. Dr. José Neuman de Souza(Orient.) II. Universidade Federal do Ceará– Pós-Graduação em Ciência da Computação(Mestrado) III. Mestre

CDD:000.0

PAULO ANTONIO LEAL REGO

FAIRCPU: UMA ARQUITETURA PARA PROVISIONAMENTO DE MÁQUINAS VIRTUAIS UTILIZANDO CARACTERÍSTICAS DE PROCESSAMENTO

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação, da Universidade Federal do Ceará, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação. Área de concentração: Ciência da Computação

Aprovada em: __/__/____

BANCA EXAMINADORA

Prof. Dr. José Neuman de Souza
Universidade Federal do Ceará - UFC
Orientador

Prof. Dr. Bruno Richard Schulze
Laboratório Nacional de Computação
Científica - LNCC

Prof. Dr. Miguel Franklin de Castro
Universidade Federal do Ceará - UFC

Prof. Dr. Paulo Henrique Mendes Maia
Grupo de Redes de Computadores Engenharia
de Software e Sistemas - GREAT

Dedico esta dissertação à toda minha família, namorada, mestres e amigos, pelo apoio incondicional, conselhos, força e amizade sem igual. Sem eles nada disto seria possível.

AGRADECIMENTOS

Meus sinceros agradecimentos:

- ao professor Dr. José Neuman de Souza, pela orientação e incentivo;
- aos professores do Departamento de Computação, pela formação, dedicação e apoio;
- aos professores do GREat, LSBD e ComCiDis, pelo suporte em momentos dessa jornada;
- aos meus pais, pela formação ética e moral, amor e apoio em todos os momentos;
- aos meus irmãos, pelo amor incondicional e companheirismo;
- à toda minha família, exemplo de união e fonte de força;
- à minha namorada, pelo amor e carinho em todas as horas;
- a todos meus amigos e colegas da Computação da UFC e BIMO;
- à FUNCAP, pelo apoio financeiro.

“Os livros não mudam o mundo, quem muda o mundo são as pessoas. Os livros só mudam as pessoas.”

(Mário Quintana)

RESUMO

O escalonamento de recursos é um processo chave para a plataforma de computação em nuvem, que geralmente utiliza máquinas virtuais (MVs) como unidades de escalonamento. O uso de técnicas de virtualização fornece grande flexibilidade com a habilidade de instanciar várias MVs em uma mesma máquina física (MF), modificar a capacidade das MVs e migrá-las entre as MFs. As técnicas de consolidação e alocação dinâmica de MVs têm tratado o impacto da sua utilização como uma medida independente de localização. É geralmente aceito que o desempenho de uma MV será o mesmo, independentemente da MF em que ela é alocada. Esta é uma suposição razoável para um ambiente homogêneo, onde as MFs são idênticas e as MVs estão executando o mesmo sistema operacional e aplicativos. No entanto, em um ambiente de computação em nuvem, espera-se compartilhar um conjunto composto por recursos heterogêneos, onde as MFs podem variar em termos de capacidades de seus recursos e afinidades de dados. O objetivo principal deste trabalho é apresentar uma arquitetura que possibilite a padronização da representação do poder de processamento das MFs e MVs, em função de unidades de processamento (UPs), apoiando-se na limitação do uso da CPU para prover isolamento de desempenho e manter a capacidade de processamento das MVs independente da MF subjacente. Este trabalho busca suprir a necessidade de uma solução que considere a heterogeneidade das MFs presentes na infraestrutura da Nuvem e apresenta políticas de escalonamento baseadas na utilização das UPs. A arquitetura proposta, chamada FairCPU, foi implementada para trabalhar com os hipervisores KVM e Xen, e foi incorporada a uma nuvem privada, construída com o *middleware* OpenNebula, onde diversos experimentos foram realizados para avaliar a solução proposta. Os resultados comprovam a eficiência da arquitetura FairCPU em utilizar as UPs para reduzir a variabilidade no desempenho das MVs, bem como para prover uma nova maneira de representar e gerenciar o poder de processamento das MVs e MFs da infraestrutura.

Palavras-chave: Provisionamento de máquinas virtuais. Ambiente heterogêneo. Limitação do uso da CPU. Computação em nuvem. Virtualização. Escalonamento. Unidade de processamento. Variabilidade de desempenho. KVM. Xen. OpenNebula.

ABSTRACT

Resource scheduling is a key process for cloud computing platform, which generally uses virtual machines (VMs) as scheduling units. The use of virtualization techniques provides great flexibility with the ability to instantiate multiple VMs on one physical machine (PM), migrate them between the PMs and dynamically scale VM's resources. The techniques of consolidation and dynamic allocation of VMs have addressed the impact of its use as an independent measure of location. It is generally accepted that the performance of a VM will be the same regardless of which PM it is allocated. This assumption is reasonable for a homogeneous environment where the PMs are identical and the VMs are running the same operating system and applications. Nevertheless, in a cloud computing environment, we expect that a set of heterogeneous resources will be shared, where PMs will face changes both in terms of their resource capacities and as also in data affinities. The main objective of this work is to propose an architecture to standardize the representation of the processing power by using processing units (PUs). Adding to that, the limitation of CPU usage is used to provide performance isolation and maintain the VM's processing power at the same level regardless the underlying PM. The proposed solution considers the PMs heterogeneity present in the cloud infrastructure and provides scheduling policies based on PUs. The proposed architecture is called FairCPU and was implemented to work with KVM and Xen hypervisors. As study case, it was incorporated into a private cloud, built with the middleware OpenNebula, where several experiments were conducted. The results prove the efficiency of FairCPU architecture to use PUs to reduce VMs' performance variability, as well as to provide a new way to represent and manage the processing power of the infrastructure's physical and virtual machines.

Keywords: Virtual machines provisioning. Heterogeneous environment. Limit CPU usage. Cloud computing. Virtualization. Scheduling. Processing unit. Performance variability. KVM. Xen. OpenNebula.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

LISTA DE SIGLAS

API	*Application programming interface*
CAD	Computação de alto desempenho
CRM	*Customer relationship management*
DAG	*Directed acyclic graph*
DVFS	*Dynamic voltage and frequency scaling*
EC2	*Elastic Compute Cloud*
GPU	*Graphics processing unit*
FLOPS	*Floating point operations per second*
IaaS	*Infrastructure as a Service*
KVM	*Kernel-based Virtual Machine*
MF	Máquina física
MIPS	*Millions of instructions per second*
MV	Máquina virtual
NA	Não se aplica
NFS	*Network File System*
OCA	*OpenNebula Cloud API*
PaaS	*Platform as a service*
QoS	*Quality of service*
SaaS	*Software as a service*
SLA	*Service level aggrement*
SLO	*Service level objective*
SO	Sistema operacional
SOA	*Service oriented architecture*
SSH	Secure shell
VCPU	CPU virtual
VMM	*Virtual machine monitor*

API CAD CRM DAG DVFS EC2 GPU FLOPS IaaS KVM MF MIPS MV NA NFS OCA PaaS QoS SaaS SLA SLO SO SOA SSH VCPU VMM

1 INTRODUÇÃO

A exigência crescente de recursos computacionais em áreas como computação científica abriu o caminho para o desenvolvimento da computação paralela e distribuída. Depois do sucesso generalizado dos *clusters* e a adoção mais lenta das grades computacionais, o modelo de computação em nuvem é cada vez mais adotado em soluções distribuídas (EVOY; SCHULZE; GARCIA, 2011).

A computação em nuvem tornou-se a expressão da moda na indústria de tecnologia de informação (TI) (WANG et al., 2010; VOUK, 2008): os usuários e empresas estão movendo seus dados e aplicações para a nuvem e acessando-os de uma forma simples e pervasiva, sem se preocupar com onde as aplicações estão atualmente instaladas e como elas foram implantadas (ENDO et al., 2010).

Em um ambiente de nuvem, espera-se uma coleção diversa de aplicações para partilhar um conjunto de recursos heterogêneos. As aplicações podem variar significativamente em termos de necessidades de recursos, padrões de acesso e interdependências (SONNEK; CHANDRA, 2009).

O escalonamento de recursos é um processo chave para a plataforma de computação em nuvem, que geralmente utiliza máquinas virtuais (MVs) como unidades de escalonamento. O uso de técnicas de virtualização fornece grande flexibilidade com a habilidade de instanciar várias MVs em uma mesma máquina física (MF), modificar a capacidade das MVs e migrá-las entre as MFs.

A virtualização permite a criação de diversos servidores virtuais sob demanda. É nesse ponto que a computação em nuvem revoluciona, pois este fato permitiu que as empresas com grandes tarefas orientadas a lote pudessem obter resultados tão mais rápido quanto seus programas pudessem ser escalados, uma vez que seria possível utilizar 1000 servidores durante 1 hora, em vez de usar 1 servidor por 1000 horas. Tamanha elasticidade de recursos é sem precedentes na história da TI (ARMBRUST et al., 2009).

A computação em nuvem é uma promessa boa para a comunidade científica, pois promete ser uma alternativa barata para supercomputadores, sendo uma plataforma muito mais confiável do que grades, e muito mais escalável do que os *clusters* especializados. Nuvens também prometem escalabilidade imediatamente, com os únicos limites impostos por razões financeiras, em oposição aos limites físicos da adição de nós para *clusters* ou mesmo supercomputadores, ou o ônus financeiro de excesso de provisionamento de recursos. No entanto, as nuvens também levantam desafios importantes em muitas áreas ligadas à computação científica, incluindo desempenho (IOSUP et al., 2011).

O problema é que as técnicas de consolidação e alocação dinâmica de MVs têm tratado o impacto da sua utilização como uma medida independente de localização. É geralmente aceito que o desempenho de uma MV será o mesmo, independentemente da MF em que ela é alocada. Esta é uma suposição razoável para um ambiente homogêneo, onde as MFs são idênticas e as MVs estão executando o mesmo sistema operacional e aplicativos. No entanto,

em um ambiente de computação em nuvem, espera-se compartilhar um conjunto composto por recursos heterogêneos, onde as MFs podem variar em termos de capacidades de seus recursos e afinidades de dados.

Este trabalho apresenta a arquitetura FairCPU, cujo principal objetivo é padronizar a representação do poder de processamento das MFs e MVs, em função de unidades de processamento (UPs), apoiando-se na limitação do uso da CPU para prover isolamento de desempenho e manter a capacidade de processamento das MVs independente da MF subjacente. Este trabalho busca suprir a necessidade de uma solução que considere a heterogeneidade das MFs presentes na infraestrutura da nuvem.

A arquitetura é descrita e o funcionamento de seus módulos são apresentados para mostrar como lidar com a heterogeneidade de uma infraestrutura de nuvem. Como estudo caso, a FairCPU foi implementada para trabalhar com os hipervisores KVM e Xen, e com o *middleware* OpenNebula. Além disso, diversos experimentos foram realizados com diferentes *benchmarks* e aplicações para avaliar a eficiência da solução proposta.

Na próxima seção, os objetivos deste trabalho são apresentados.

1.1 Objetivos

1.1.1 Objetivo Geral

Especificar e implementar uma arquitetura que possibilite a padronização da representação do poder de processamento das MFs e MVs em função das unidades de processamento (que terão valores constantes em termos de GFLOPs, ou outra métrica), aliado à limitação do uso da CPU para prover isolamento de desempenho (*performance isolation*[1]) e manter a capacidade de processamento independente da MF em que a MV tenha sido alocada.

1.1.2 Objetivos Específicos

1. Avaliar a utilização de programas para limitar o uso da CPU destinada às MVs;
2. Especificar a arquitetura FairCPU;
3. Implementar a arquitetura FairCPU e integrar com um *middleware*[2] de código aberto;
4. Desenvolver políticas de escalonamento baseadas em unidades de processamento;
5. Avaliar a proposta de arquitetura ao comparar com sistemas que não utilizam UPs;
6. Disponibilizar para a comunidade todos os artefatos desenvolvidos neste trabalho.

[1] É uma medida de quão bem as MVs visitantes estão protegidas do intenso consumo de recursos das outras MVs (DESHANE et al., 2008).

[2] Podemos utilizar o OpenNebula, devido ao nosso conhecimento da ferramenta e a infraestrutura já montada no nosso laboratório.

1.2 Organização do Trabalho

O restante deste trabalho está organizado da seguinte forma. O Capítulo 2 apresenta uma visão geral sobre os principais temas relacionados ao presente trabalho, como computação em nuvem, virtualização e escalonamento de recursos.

O Capítulo 3 apresenta a motivação e detalha o problema abordado, apresentando inclusive um experimento motivacional. No Capítulo 4, a arquitetura FairCPU é apresentada, assim como todos seus módulos e importantes detalhes de implementação. O Capítulo 5 apresenta diversos experimentos que foram realizados para comprovar a eficiência da solução proposta, bem como seus pontos fracos.

Os trabalhos relacionados são discutidos no Capítulo 6, enquanto o Capítulo 7 apresenta as conclusões e os trabalhos futuros.

2 FUNDAMENTOS CONCEITUAIS

Este capítulo destina-se à introdução dos fundamentos conceituais sobre os temas computação em nuvem, virtualização e escalonamento de recursos, que estão diretamente relacionados a este trabalho.

2.1 Computação em Nuvem

Com o rápido desenvolvimento das tecnologias de processamento e armazenamento, e o grande sucesso da Internet, os recursos computacionais tornaram-se mais baratos, mais poderosos e mais ubiquamente disponíveis que antes. Esta nova tendência tecnológica tornou possível a existência desse novo modelo de computação, chamado computação em nuvem, onde os recursos são providos sob demanda através da Internet (ZHANG; CHENG; BOUTABA, 2010).

A computação em nuvem tornou-se a expressão da moda na indústria de TI (WANG et al., 2010; VOUK, 2008) e tem o objetivo de proporcionar serviços sob demanda com pagamento baseado no uso (*pay-per-use*). Tendências anteriores à computação em nuvem foram limitadas a uma determinada classe de usuários ou focadas em tornar disponível uma demanda específica de recursos de TI (BUYYA et al., 2009). Já a computação em nuvem pretende ser global e prover serviços para as massas que vão desde o usuário final, que hospeda seus documentos pessoais na Internet, até empresas que terceirizam toda infraestrutura de TI para outras empresas (SOUSA et al., 2011). Decerto, usuários e empresas estão movendo seus dados e aplicações para a nuvem e acessando-os de uma forma simples e pervasiva, sem se preocupar com onde as aplicações estão atualmente instaladas e como elas foram implantadas (ENDO et al., 2010).

Este parece ser um novo cenário de centralização do processamento. Similar ocorreu por volta de 50 anos atrás com a computação em servidores de tempo compartilhado para vários usuários. Até 20 anos atrás, quando os computadores pessoais chegaram até nós, dados e programas eram, em sua maioria, localizados em recursos locais. Certamente o atual paradigma da computação em nuvem não é uma repetição da história, uma vez que, 50 anos atrás, nós tivemos que adotar os servidores de tempo compartilhado devido à limitação dos recursos computacionais. Hoje, a computação em nuvem entra em moda devido à necessidade de construir complexas infraestruturas de TI (WANG et al., 2010).

Wang et al. (2010) definem computação em nuvem como um conjunto de serviços sobre a rede, provendo escalabilidade, garantia de qualidade de serviço (QoS) e infraestrutura de computação sob demanda a baixo custo, que pode ser acessado de maneira simples e abrangente. Rimal, Choi e Lumb (2009) complementam a definição de computação em nuvem, ao afirmar que o conceito aborda a nova fase evolucionária da computação distribuída, cujo objetivo é fazer o melhor uso dos recursos distribuídos, organizando-os com o intuito de alcançar maior taxa de transferência, economizar energia e ser capaz de resolver problemas computacionais de larga escala.

Existem diversas definições e percepções para esse novo modelo de computação, talvez por isso Vaquero et al. (2008) catalogaram 22 dessas definições. A principal razão para tantas percepções é o fato de que a computação em nuvem, ao contrário de outros termos técnicos, não é uma tecnologia nova, mas sim um novo modelo de operações que reúne um conjunto de tecnologias já existentes para fazer negócios de uma maneira diferente (ZHANG; CHENG; BOUTABA, 2010).

Algumas das tecnologias relacionadas à computação em nuvem são:

- Sistemas de armazenamento distribuído;
- Virtualização;
- Web 2.0;
- Orquestração de fluxo de serviço e fluxo de trabalho;
- *Web service* e arquitetura orientada a serviços (SOA).

A virtualização é a principal das tecnologias envolvidas. Ela forma a base da computação em nuvem, uma vez que fornece a capacidade de reunir recursos de computação de um conjunto de servidores e dinamicamente atribuir ou reatribuir recursos virtuais para aplicações sob demanda (ZHANG; CHENG; BOUTABA, 2010).

A infraestrutura do ambiente de computação em nuvem normalmente é composta por um grande número de MFs, conectadas por meio de uma rede como ilustra a Figura 2.1. As MFs geralmente tem as mesmas configurações de *software*, mas pode haver variação na capacidade de *hardware* em termos de CPU, memória e armazenamento em disco (SOROR et al., 2010). Dentro de cada MF é executado um número variável de MVs, de acordo com a capacidade do *hardware* disponível na MF. Os dados são persistidos, geralmente em sistemas de armazenamento distribuído.

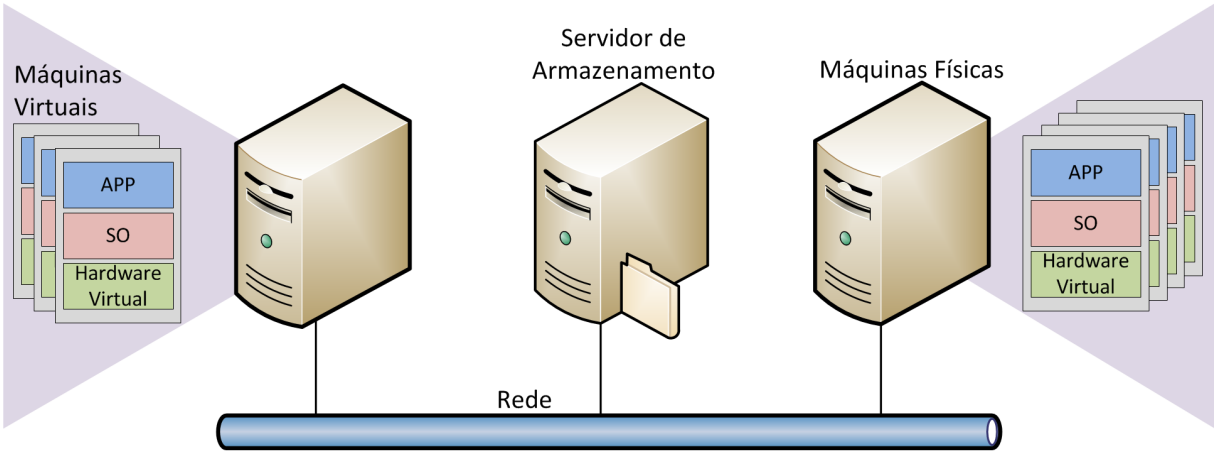


Figura 2.1: Ambiente de computação em nuvem

O modelo de computação em nuvem é composto, tipicamente, por cinco características essenciais, três modelos de serviços e quatro modelos de implantação da nuvem (MELL; GRANCE, 2009). As características são (SOUSA et al., 2011; VERDI et al., 2010):

Serviço sob demanda: o usuário pode adquirir unilateralmente recurso computacional na medida em que necessite e sem precisar de interação humana com os provedores de serviço.

Amplo acesso: os recursos computacionais são disponibilizados por meio da Internet e acessados através de mecanismos padronizados que possibilitam o uso por dispositivos móveis, computadores, etc.

***Pooling* de recursos:** os recursos computacionais do provedor são utilizados para servir múltiplos usuários usando um modelo multi-inquilino (*multi-tenant*), com diferentes recursos físicos e virtuais, dinamicamente atribuídos e ajustados de acordo com a demanda dos usuários. Estes usuários não precisam ter conhecimento da localização física dos recursos computacionais, podendo somente especificar a localização em um nível mais alto de abstração, tais como o país, estado ou centro de dados.

Elasticidade: recursos podem ser adquiridos de forma rápida e elástica, em alguns casos automaticamente, caso haja a necessidade de escalar com o aumento da demanda, e liberados, na retração dessa demanda. Para os usuários, os recursos disponíveis para uso parecem ser ilimitados e podem ser adquiridos em qualquer quantidade e a qualquer momento.

Medição dos serviços: os sistemas de gerenciamento utilizados para computação em nuvem controlam e monitoram automaticamente os recursos para cada tipo de serviço (armazenamento, processamento e largura de banda). Este monitoramento do uso dos recursos deve ser transparente para o provedor do serviço, assim como, para o consumidor do serviço utilizado.

O ambiente de computação em nuvem é composto de três modelos de serviços (VAQUERO et al., 2008; RIMAL; CHOI; LUMB, 2009), como ilustra a Figura 2.2. Estes modelos são importantes, pois eles definem um padrão arquitetural para soluções de computação em nuvem. Os modelos são (SOUSA et al., 2011; VERDI et al., 2010):

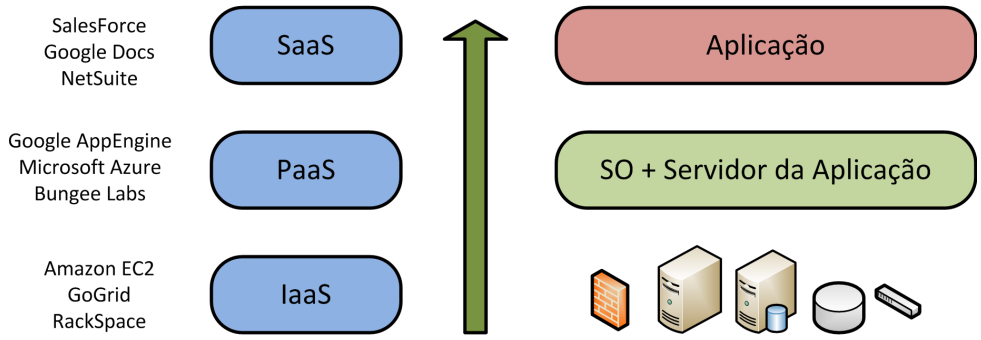


Figura 2.2: Ilustração com exemplos dos três modelos de serviços de computação em nuvem.

***Software* como um Serviço (*Software as a Service* - SaaS):** este modelo proporciona sistemas de *software* com propósitos específicos, que são disponíveis para os usuários por meio da Internet e acessíveis a partir de aplicações como o navegador Web. No SaaS, o usuário não administra ou controla a infraestrutura subjacente, incluindo rede, servidores, sistema operacional, armazenamento ou mesmo as características individuais da aplicação,

exceto configurações específicas. Como exemplos de SaaS podemos destacar os serviços de Gestão de Relacionamento com o Cliente (*Customer Relationship Management* - CRM) da Salesforce (SALESFORCE, 2012) e o Google Docs.

Plataforma como um Serviço (*Platform as a Service* - PaaS): este modelo fornece sistema operacional, linguagens de programação e ambientes de desenvolvimento para as aplicações, auxiliando a implementação de sistemas de *software*. Assim como no SaaS, o usuário não administra ou controla a infraestrutura subjacente, mas tem controle sobre as aplicações implantadas e, possivelmente, as configurações de aplicações hospedadas nesta infraestrutura. Google App Engine (CIURANA, 2009) e Microsoft Azure (AZURE, 2012) são exemplos de PaaS.

Infraestrutura como um Serviço (*Infrastructure as a Service* - IaaS): este modelo torna mais fácil e acessível o fornecimento de recursos, tais como servidores, rede, armazenamento e outros recursos de computação fundamentais para construir um ambiente de aplicação sob demanda, que podem incluir sistemas operacionais e aplicativos. Em geral, o usuário não administra ou controla a infraestrutura da nuvem, mas tem controle sobre os sistemas operacionais, armazenamento, aplicativos implantados e, eventualmente, seleciona componentes de rede, tais como *firewalls*. Amazon Elastic Cloud Computing (EC2), GoGrid (GOGRID, 2012) e RackSpace (RACKSPACE, 2012) são exemplos de provedores de IaaS.

Apesar de se pensar geralmente em aplicações interativas do tipo SaaS, a computação em nuvem vem sendo bastante utilizada para fazer processamento paralelo em lote, o que possibilita a analise de vários terabytes de dados de forma mais rápida, ao executar processamentos em paralelo (ARMBRUST et al., 2009).

Chaisiri, Lee e Niyato (2009) afirmam que o modelo de IaaS foi identificado como o modelo de computação em nuvem com mais potencial, pois é através deste que é possível gerenciar os recursos necessários para a execução de aplicações. É justamente nessa camada que nossa solução trabalha.

Quanto ao acesso e à disponibilidade, há diferentes tipos de modelos de implantação para os ambientes de computação em nuvem. A restrição ou abertura de acesso depende do processo de negócios, do tipo de informação e do nível de visão desejado. Os modelos de implantação são (SOUSA et al., 2011; VERDI et al., 2010):

Nuvem privada (*private cloud*): compreende uma infraestrutura de nuvem que é utilizada exclusivamente por uma organização, sendo esta nuvem local ou remota e administrada pela própria empresa ou por terceiros.

Nuvem pública (*public cloud*): a infraestrutura de nuvem é disponibilizada para o público em geral através do modelo de pagamento baseado no uso. São tipicamente oferecidas por companhias que possuem grandes capacidades de armazenamento e processamento, como a Amazon, Microsoft, etc.

Nuvem comunidade (*community cloud*): fornece uma infraestrutura compartilhada por uma comunidade de organizações com interesses em comum.

Nuvem híbrida (*hybrid cloud*): compreende uma infraestrutura que é composta de duas ou mais nuvens, que podem ser do tipo privada, pública ou comunidade e que continuam a ser entidades únicas, mas conectadas por meio de tecnologia proprietária ou padronizada que permite a portabilidade de dados e aplicações.

Após analisar as definições apresentadas, Verdi et al. (2010) afirmam que é possível destacar três novos aspectos em relação ao *hardware*, introduzidos em computação em nuvem. São eles:

- A ilusão de recurso computacional infinito disponível sob-demanda;
- A eliminação de um comprometimento antecipado por parte do usuário;
- A capacidade de alocar e pagar por recursos usando uma granularidade de horas.

Essa elasticidade para obter e liberar recursos é um dos aspectos chaves da computação em nuvem, sendo possível graças à virtualização.

2.2 Virtualização

A virtualização tem sido definida como a abstração dos recursos do computador ou como uma técnica para esconder as características físicas dos recursos computacionais e pode ser feita de diversas formas:

Virtualização de *desktops*: é a virtualização que trata da configuração dos *desktops* de usuários finais em uma infraestrutura virtual, onde os sistemas operacionais e as aplicações passam a ser executados em um *datacenter*, sob a forma de uma MV. Cada usuário pode customizar seu ambiente de trabalho e acessá-lo de diferentes locais, utilizando qualquer estação de trabalho na rede (IBM, 2007).

Virtualização de armazenamento: é um conceito referente à abstração entre o armazenamento lógico e o armazenamento físico. A ideia desse tipo de virtualização é fazer com que diversos discos físicos sejam vistos como um conjunto homogêneo de recursos de armazenamento. Esta separação permite que os administradores de sistemas possam aumentar a flexibilidade no gerenciamento de usuários e cotas de disco, uma vez que as complexidades dos discos físicos são escondidas.

Virtualização de redes: é a virtualização que permite a criação de ambientes de rede separados para cada grupo de usuários, apesar de compartilhar a mesma rede física. Para os usuário finais, é como se eles estivessem acessando uma rede própria, com recursos dedicados e políticas de segurança independentes. Neste tipo de virtualização é possível criar diversos dispositivos de rede, como *switches* virtuais (ZHOU, 2010), como ilustra a Figura 2.3.

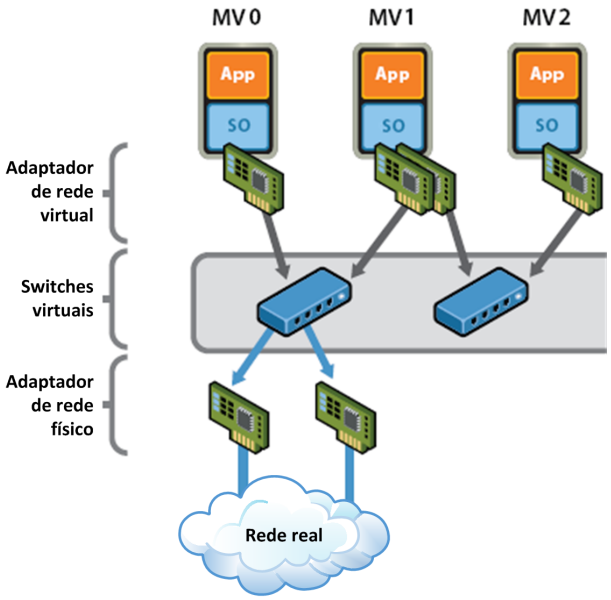


Figura 2.3: Virtualização de *redes*.

Virtualização de servidores: é o tipo de virtualização mais comum. Esse tipo de virtualização permite que uma MF seja particionada para executar múltiplos ambientes virtuais, na forma de MVs.

A virtualização é uma tecnologia-chave da plataforma de computação em nuvem, onde as aplicações são encapsuladas dentro de MVs e estas são mapeadas dinamicamente para um conjunto de MFs (SONNEK; CHANDRA, 2009). Os serviços em execução nas MVs são executados separadamente de todos os processos das MFs, incluindo o sistema operacional destas. Esta separação traz diversos benefícios como segurança e portabilidade (CHEN; NOBLE, 2001).

A utilização de técnicas de virtualização pode trazer outros benefícios, como (VMWARE, 2011):

- redução dos custos operacionais;
- redução do tempo gasto em tarefas administrativas de TI;
- facilidade para fazer *backup* e proteção de dados;
- consolidação de servidores;
- aumento da disponibilidade de aplicativos;
- facilidade para recuperação de falhas.

Uma MV provê uma abstração virtualizada de uma MF. O *software* que faz com que um servidor suporte a implantação de MVs, tipicamente chamado de monitor de máquina virtual (VMM - *Virtual Machine Monitor*) ou hipervisor (*hypervisor*) (SMITH; NAIR, 2005), é

responsável por suportar esta abstração e interceptar e emular algumas instruções emitidas pelas máquinas hospedeiras. Um hipervisor provê uma interface que permite ao usuário inicializar, pausar, serializar e desligar múltiplas MVs (KEAHEY et al., 2007).

Existem diversos tipos de hipervisores, como Xen, VMWare ESX, Kernel Virtual Machine (KVM), VirtualBox, Microsoft Hyper-V, oVirt e RTS Hypervisor. Os mais populares são Xen e KVM, que são de código aberto, VirtualBox, que tem uma versão de código aberto e uma comercial, e VMware, que é comercial. Estes programas provêm um *framework* que permite que as MVs sejam executadas (SEMPOLINSKI; THAIN, 2010). Neste trabalho iremos abordar mais a utilização do Xen e KVM, que são de código aberto e amplamente utilizados.

Assim como existem diversos hipervisores, existem diferentes maneiras de fazer virtualização. A principal diferença entre elas é a maneira como as instruções privilegiadas das MVs chegam de fato ao *hardware*. Sistemas operacionais x86 são projetados para funcionar diretamente sobre o *hardware*, de modo que, naturalmente, eles assumem que têm o controle total sobre o *hardware*. Conforme mostrado na Figura 2.4, a arquitetura x86 oferece quatro níveis de privilégio, conhecidos como Anel 0, 1, 2 e 3, para sistemas operacionais e aplicativos poderem gerenciar o acesso ao *hardware* do computador.

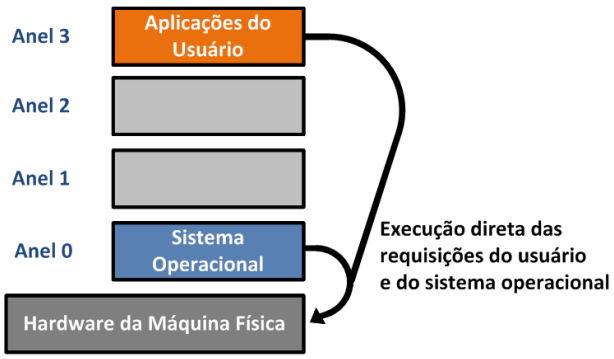


Figura 2.4: Níveis de privilégio da arquitetura x86 sem virtualização. Fonte: (VMWARE, 2007).

Enquanto as aplicações em nível de usuário normalmente são executadas no Anel 3, o sistema operacional precisa ter acesso direto à memória e *hardware*, e deve executar suas instruções privilegiadas no Anel 0. A virtualização da arquitetura x86 requer uma camada de virtualização sob o SO para criar e gerenciar as MVs que acessam recursos compartilhados. O grande problema é que algumas instruções sensíveis não podem efetivamente ser virtualizadas, pois têm semânticas diferentes quando não são executadas no Anel 0. A dificuldade em capturar e traduzir estes pedidos de instrução sensíveis e privilegiadas, em tempo de execução, foi o desafio que originalmente fez a virtualização da arquitetura x86 parecer impossível (VMWARE, 2007).

As diferentes maneiras de fazer virtualização são apresentadas a seguir (VMWARE, 2007):

Virtualização total (*full virtualization*): este tipo de virtualização fornece uma simulação completa do *hardware* subjacente através da emulação de *hardware*. Dispositivos de *hardware*

artificiais são criados com tudo o que é preciso para executar um SO, sem a necessidade de modificar o *kernel* do SO visitante. Geralmente, utiliza-se uma combinação de tradução binária (como pode ser visto na Figura 2.5) e técnicas de execução direta para executar as chamadas do sistema. Essas chamadas são interceptadas pelo hipervisor, que as mapeia para o *hardware* real subjacente, enquanto parte do código do nível do usuário pode ser executado diretamente no processador para obter um melhor desempenho. O hipervisor garante total independência e autonomia entre as MVs em execução na mesma MF. Com a virtualização total, o SO visitante não tem conhecimento de que está sendo executado em *hardware* virtualizado e, por isso mesmo, não requer nenhuma modificação. Com este tipo de virtualização é possível executar MVs com SO Windows em MFs com SO Linux, por exemplo. Os hipervisores VMWare e Virtual Box utilizaram esse tipo de virtualização.

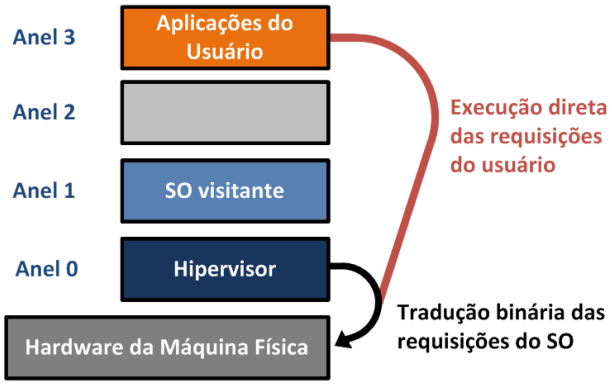


Figura 2.5: Abordagem com tradução binária para prover virtualização total na arquitetura X86. Fonte: (VMWARE, 2007).

Paravirtualização (*paravirtualization*): neste tipo de virtualização, o *kernel* do SO visitante é modificado especificamente para executar no hipervisor. Isto envolve a substituição de quaisquer operações privilegiadas, que só seriam executadas no Anel 0 da CPU, por chamadas para o hipervisor (como pode ser visto na Figura 2.6), conhecidas como hiperchamadas (*hypercalls*). O hipervisor, por sua vez executa a tarefa em nome do *kernel* da MV e também fornece interfaces de hiperchamada para outras operações críticas do *kernel*, tais como gestão de memória ou gestão de interrupções. A paravirtualização tenta corrigir os problemas da virtualização total permitindo que os SOs visitantes tenham acesso direto ao *hardware* subjacente. Na paravirtualização, o SO visitante sabe que está sendo executado em *hardware* virtualizado, diferente da virtualização total, onde as chamadas críticas são interceptadas e traduzidas usando tradução binária.

Virtualização ao nível do sistema operacional (*OS-level virtualization*): este tipo de virtualização é baseado na criação de *containers* ou partições isoladas em uma única MF. Com esta técnica, instala-se a camada de *software* de virtualização em cima do SO e todos as MVs executam sobre essa camada, usando o mesmo SO do sistema hospedeiro, mas tendo cada uma seus próprios recursos e funcionando isolada das outras MVs. Nesta virtualização cada MV tem seu próprio sistema de arquivos mas partilham o *kernel* do SO hospedeiro e o hipervisor tem uma funcionalidade muito limitada, contando com o SO

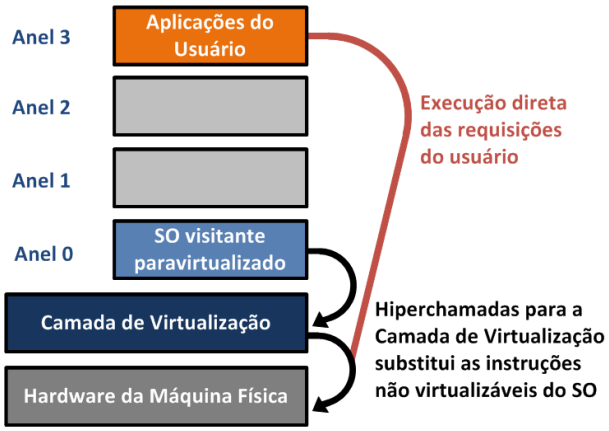


Figura 2.6: Abordagem com paravirtualização na arquitetura X86. Fonte: (VMWARE, 2007).

hospedeiro para o escalonamento de CPU e gestão de memória. Alguns dos hipervisores que utilizam esse tipo de virtualização são: OpenVZ, Virtuozzo ou Solaris Zones.

Virtualização assistida por *hardware* (*hardware-assisted virtualization*): este tipo de virtualização utiliza recursos de virtualização incorporados nas últimas gerações de CPUs da Intel e AMD. As tecnologias conhecidas como Intel VT e AMD-V oferecem extensões necessárias para executar MVs com SO não modificado, sem as desvantagens inerentes à emulação de CPU da virtualização total, como a necessidade de tradução binária. Estes novos processadores fornecem um modo de privilégio adicional (também chamado de Anel -1) em que o hipervisor pode operar, como pode ser visto na Figura 2.7. Com a virtualização assistida por *hardware*, o hipervisor pode virtualizar eficientemente todo o conjunto de instruções x86 ao lidar com essas instruções críticas usando um modelo clássico por *hardware*, em vez de por *software*. Os hipervisores que suportam esta tecnologia podem funcionar no Anel -1 e os SOs visitantes podem utilizar a CPU no Anel 0, como fariam normalmente se estivessem a ser executados numa MF. Isto permite virtualizar SOs visitantes sem nenhuma modificação.

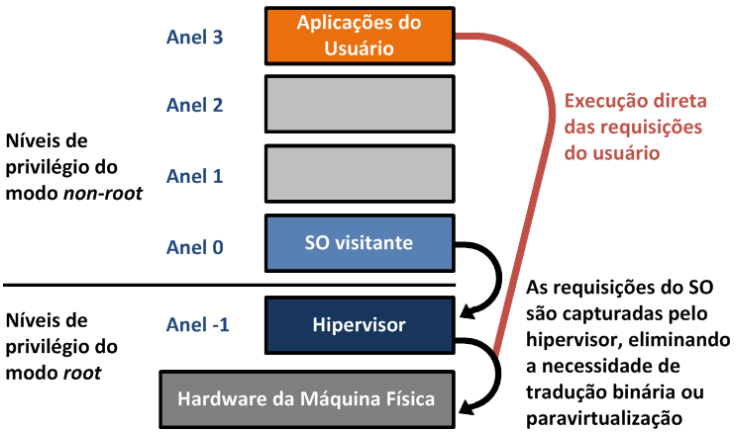


Figura 2.7: Abordagem com virtualização assistida por *hardware* na arquitetura X86. Fonte: (VMWARE, 2007).

Como existem diferentes técnicas e maneiras de fazer virtualização, é comum encontrar uma grande variabilidade entre o desempenho das MVs quando executadas com diferentes hipervisores, principalmente desempenho de CPU e I/O de rede, que são prejudicados quando executados em dispositivos completamente emulados. Diversos trabalhos científicos tem avaliado o desempenho de diferentes técnicas de virtualização, mas acredita-se que com o tempo as técnicas vão ser melhoradas e diversas otimizações levarão a um desempenho mais consistente e uniforme (IBM, 2007). Regola e Ducom (2010) afirmam que a perda de desempenho de CPU foi praticamente eliminada nas virtualizações ao nível do sistema operacional, total e paravirtualização, faltando apenas melhorias com relação ao desempenho de I/O.

A virtualização permite a criação de diversas MVs sob demanda. É nesse ponto que a computação em nuvem revoluciona e influencia as técnicas de programação, pois este fato permitiu que as empresas com grandes tarefas orientadas a lote pudessem obter resultados tão mais rápido quanto seus programas pudessem ser escalados (ARMBRUST et al., 2009).

Outra prova da importância da virtualização está no fato de algumas das principais empresas da indústria de TI (IBM, Intel, Hewllet-Packard, Oracle, Red Hat) estarem mostrando interesse na tecnologia de virtualização. Muitas empresas clássicas da área de armazenamento como a EMC, NetApp, IBM e Hitachi também estão se envolvendo. Além disso, o mercado de virtualização de rede está repleto de atividade (VOUK, 2008).

2.3 Escalonamento de Recursos

A necessidade de recursos computacionais tem se tornado um requisito fundamental tanto para a academia quanto para a indústria. Em muitos casos, a necessidade é transiente: um usuário pode precisar de recursos computacionais, por algumas horas, para realizar uma tarefa bem definida. Por exemplo, um cientista pode precisar de centenas de computadores para executar uma simulação durante 1 dia, um professor pode precisar de dezenas de computadores para uma aula prática de 2 horas, uma empresa pode precisar de 50 computadores para seu site de comércio eletrônico nos horários de pico, enquanto 10 são suficientes durante o resto do dia.

Esses cenários trazem o problema de como prover recursos computacionais compartilhados de maneira eficiente. Esse problema tem sido estudado há décadas, o que resultou em diversas abordagens que tendem a ser altamente especializadas para um cenário de uso específico (SOTOMAYOR, 2010).

Existem diversos escalonadores de recursos (TANNENBAUM et al., 2002; HENDERSON, 1995) para ambientes distribuídos, muitos foram propostos ainda durante o estudo da virtualização sobre grades computacionais. Com o advento do paradigma de computação em nuvem, muitas das soluções propostas têm sido remodeladas e diversas outras vêm sendo propostas.

As duas principais abordagens para os escalonadores são:

Baseada em tarefas (*job-based*): na academia e, até certo ponto, na indústria, a provisão de recursos tem sido geralmente amarrada à execução de tarefas e muitos escalonadores de

tarefas tem sido desenvolvidos nos últimos anos. Estes escalonadores devem considerar as dependências das tarefas e a sua representação na forma de *workflows*. Esses escalonadores geralmente dependem de filas para priorizar o acesso aos recursos e usam *backfilling* (SRINIVASAN et al., 2002) para otimizar a ordenação destas filas. O tempo de execução, *deadline* e dependências da tarefa são os parâmetros comumente usados pelo escalonador para definir em que máquinas a tarefa será executada. A Figura 2.8 apresenta uma entrada comum para essa abordagem, onde as tarefas e suas dependências formam um DAG (*Directed acyclic graph*).

Baseada em máquinas virtuais (*VM-based*): as nuvens do modelo IaaS geralmente usam MVs como unidades de escalonamento para serem alocadas em recursos físicos heterogêneos (ZHONG; TAO; ZHANG, 2010). As MVs são um veículo atraente para o gerenciamento de recursos porque elas podem ser usadas para prover *hardware*, *software* e disponibilidade (SOTOMAYOR, 2010). Para instanciar uma MV é preciso definir quanto de CPU, memória e disco ela terá, por isso a alocação é geralmente feita baseada na quantidade de recursos requisitados. Além disso, outros parâmetros podem ser considerados, como a sobrecarga para distribuir e preparar a MV (SOTOMAYOR, 2010). A Figura 2.9 ilustra bem essa abordagem, onde o escalonador deve alocar as MVs nas MFs levando em consideração os recursos requisitados para cada MV.

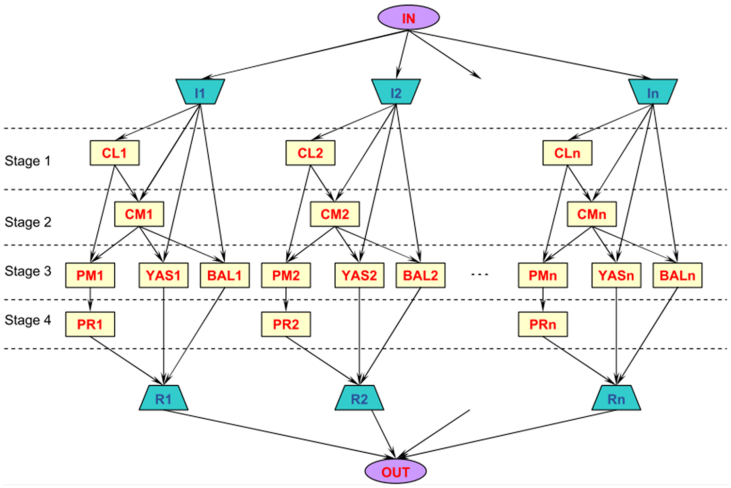


Figura 2.8: Exemplo de DAG, que são comumente utilizados como entrada para os escalonadores baseados em tarefas. Neste exemplo a aplicação tem n entradas. Fonte: (HUU; MONTAGNAT, 2010).

Existem outros tipos de escalonadores, com abordagens que consideram acordos de nível de serviço (*Service Level Agreement* - SLA) (STAGE; SETZER, 2009) e outras que misturam os dois tipos citados anteriormente (ao considerar a execução de tarefas em MVs), mas não serão diretamente explorados nesse trabalho.

Segundo Zhang, Cheng e Boutaba (2010), a computação em nuvem se aproveita das tecnologias de virtualização para alcançar o objetivo de prover recursos computacionais como deseja a computação utilitária. O provedor de infraestrutura pode se aproveitar das tecnolo-

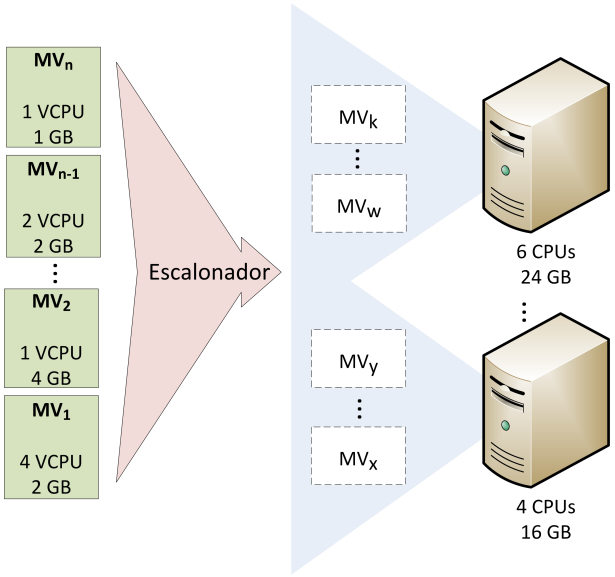


Figura 2.9: Ilustração do escalonamento baseado em MVs, onde os recursos requisitados para cada MV e os recursos disponíveis nas MFs devem ser considerados.

gias de migração de MVs para atingir um alto grau de consolidação dos servidores, e assim, maximizar a utilização de recursos enquanto diminui os custos com energia e refrigeração.

Para que este alto grau de consolidação dos servidores seja alcançado, o escalonador da infraestrutura desempenha um papel muito importante. Ele é o responsável pela alocação das MVs nas MFs. Cabe a ele definir onde as MVs devem ser executadas, e caso necessário, escolher pra onde elas devem ser migradas, e com isso, fazer melhor uso dos recursos distribuídos. O escalonamento de recursos é, então, um processo chave para as nuvens do modelo IaaS.

Grit et al. (2006) afirmam que um dos principais desafios de hoje é desenvolver políticas concretas para alocação de recursos adaptáveis e confiáveis sob demanda. Isto porque há diversos desafios arquiteturais e algorítmicos quando tratamos de políticas de gerenciamento de recursos. O primeiro deles é o fato de o gerenciamento de recursos envolver projeções e problemas de otimização que são NP-Difícil em sua forma geral. Sendo necessária a adoção de heurísticas que visam objetivos específicos. Em segundo lugar, temos o fato da alocação de recursos ser dinâmica, o que faz com que as políticas implementadas precisem se adaptar às mudanças de carga de trabalho e novas requisições em tempo real.

Segundo Khatua, Ghosh e Mukherjee (2010), na pesquisa de computação de alto desempenho (CAD), a utilização ótima dos recursos tem sido sempre considerada como um dos principais desafios. Certos problemas de utilização de recursos surgem mesmo em ambientes simples, como estações de trabalho individuais. Eles são ainda mais agravados em ambientes complexos, especialmente naqueles que são dinâmicos e heterogêneos. Algumas soluções foram propostas na literatura para ambientes tradicionais, como *clusters*. Estas soluções são baseadas principalmente em técnicas de balanceamento dinâmico de carga. No entanto, não têm sido realizados muitos trabalhos para ambientes virtuais a este respeito.

3 PROBLEMA E MOTIVAÇÃO

As técnicas de consolidação e alocação dinâmica de MVs têm tratado o impacto da sua utilização como uma medida independente de localização. É geralmente aceito que o desempenho de uma MV será o mesmo, independentemente da MF em que ela é alocada. Esta é uma suposição razoável para um ambiente homogêneo, onde as MFs são idênticas e as MVs estão executando o mesmo sistema operacional e aplicativos. No entanto, em um ambiente de computação em nuvem, espera-se o compartilhamento de um conjunto composto por recursos heterogêneos, onde as MFs podem variar em termos de capacidades de seus recursos e afinidades de dados (SONNEK; CHANDRA, 2009).

O desempenho das MVs está diretamente relacionado à MF em que ela está alocada. Como o ambiente de computação em nuvem é bastante heterogêneo, podemos ter MFs com diferentes quantidades de CPU, e estas podem possuir poder de processamento diferente. Essa diferença entre as CPUs tem influência direta no desempenho das MVs, e, consequentemente, nas aplicações que estiverem encapsuladas nelas.

A maioria dos *middleware* de computação em nuvem utilizam *templates*, como o da Figura 3.1, para personalizar a criação das MVs. Nestes *templates* são definidas diversas informações necessárias para a instanciação da MV, como a quantidade de VCPUs e memória. Estes dados são utilizados pelo escalonador da infraestrutura para definir onde as MVs serão alocadas.

```
NAME     = "MV_exemplo"
VCPU     = 2
MEMORY = 2048

DISK   = [ image = "DebianSqueeze" ]

DISK   = [ type       = swap,
           size       = 1024,
           readonly = "no" ]

NIC    = [ network = "RedePrivada" ]

REQUIREMENTS = "CPUSPEED > 1000"
```

Figura 3.1: Exemplo de *template* para criação de MV.

Grandes empresas como a Amazon (AMAZON, 2012b) servem uma diversidade de *templates*, com diferentes possibilidades de configuração para as MVs. Ao utilizar os serviços da Amazon, o cliente deve escolher o tamanho da instância desejada (*Small, Medium, Large*, etc) e cada uma delas tem uma configuração de memória e CPU específica. Como eles tem uma infraestrutura gigantesca podem garantir que sempre haverá uma MF livre para prover a configuração escolhida pelo cliente. Em um caso hipotético, pode acontecer do cliente escolher uma MV do tipo *Small*, por exemplo, e as MFs com configurações mais simples, que são usadas para alocar esse tipo de MV, não estarem disponíveis. Neste caso, a MV seria alocada numa MF mais potente, o que resultaria em um melhor desempenho para as aplicações do cliente, o

que não parece justo com os outros clientes.

A solução encontrada pelas grandes empresas foi apresentar para os clientes qual será o desempenho mínimo das MVs, ao apresentar os valores nivelados por baixo. Voltando ao caso citado, se a empresa oferecer para o *template Small* uma CPU com 1,8 GHz, caso não seja possível alocar a MV numa MF com CPU de 1,8 GHz, a empresa pode alocar numa que tenha CPU de 2,2 GHz. O desempenho superior fica de bônus para o cliente.

No caso de nuvens privadas, onde existem menos MFs, porém a grande variedade dos tipos de processadores também existe, pode-se utilizar a estratégia que iremos propor para representar o poder de processamento das máquinas da infraestrutura e utilizar melhor os recursos que geralmente são bem limitados nesse tipo de nuvem. Existem diversos *middleware* de código aberto para computação em nuvem, como Eucalyptus (EUCALYPTUS, 2012) e OpenNebula (OPENNEBULA, 2012), porém nenhum deles utiliza qualquer estratégia para representar e alocar o poder de processamento das MVs, de uma maneira que garanta o desempenho de processamento independente da MF em que ela for alocada.

O Eucalyptus utiliza a mesma definição de *templates* da Amazon, onde as MVs precisam ter as configurações de uma categoria predefinida, porém, ao criar duas MVs do tipo *Small*, é possível que elas sejam alocadas em MFs diferentes. Dessa forma, as MVs podem ter poder de processamento diferentes, uma vez que estes serão diretamente relacionados ao processador existente na MF.

O mesmo acontece no OpenNebula. Na Figura 3.2 é possível ver como o poder de processamento é representado nesse *middleware*. Não é possível fazer qualquer relação entre as MFs presentes na infraestrutura. O *host08* consta com 1200 de ACPU (total de CPU disponível para alocação), enquanto o *host03* com 400, mas não quer dizer que o *host08* tem o triplo do poder de processamento do *host03*. Se fossem criadas 2 MVs com mesma configuração (1 VCPU e 512 de memória, por exemplo), cada MV receberia 100 da ACPU, mas não quer dizer que elas terão o mesmo desempenho, pois o poder de processamento seria dependente do processador da MF.

```
pesquisa@controlador:~$ onehost list
  ID NAME            CLUSTER   RVM   TCPU   FCPU   ACPU    TMEM    FMEM STAT
   0 host01          default     1    400    397    200   23.6G   22.6G on
   1 host02          default     1    400    400    200    3.9G    3.3G on
   2 host03          default     0    400    400    400    3.9G    3.6G on
   3 host04          default     1    400    400    200    7.8G    7.2G on
   4 host05          default     0   1200   1200   1200   15.7G   15.4G on
   5 host06          default     0   1200   1200   1200   15.7G   15.4G on
   6 host07          default     2   1200   1200    800   15.7G   14.4G on
   7 host08          default     0   1200   1200   1200   15.7G   15.4G on
   9 host09          default     0   1200   1200   1200     16G      1G on
```

Figura 3.2: Representação dos dados no *middleware* OpenNebula.

Para exemplificar esse problema, um experimento motivacional foi realizado. O *benchmark* Intel Linpack foi executado em uma máquina virtual com 4 VCPUs (CPUs Virtuais), 2 GB de memória e sistema operacional Ubuntu Server 10.10, 64 bits e *kernel* 2.6.35-25. Esta MV foi executada em três MFs diferentes, Servidor A, B e C, cujas configurações podem ser

vistas na Tabela 3.1.

Servidor A	Servidor B	Servidor C
Intel Corei5-750 @ 2,66 GHz sem Hyper-Threading	Intel Corei7-930 @ 2,80 GHz com Hyper-Threading	Intel Xeon X3430 @ 2,40 GHz sem Hyper-Threading
4 GB memória	24 GB memória	8 GB memória
Ubuntu Server 11.04 64 bits, kernel 2.6.38-8-server		
Hipervisor KVM		

Tabela 3.1: Configuração das máquinas físicas para o experimento motivacional.

O resultado deste experimento foi um poder de processamento de 27,08 GFLOPS para a MV que foi alocada no Servidor A, 24,02 GFLOPS para a MV alocada no Servidor C e 15,07 GFLOPS para a MV que foi alocada no Servidor B, o que mostra a grande variabilidade entre os resultados, causada pela diferença entre as MFs. O resultado desse experimento pode ser visto na Figura 3.3.

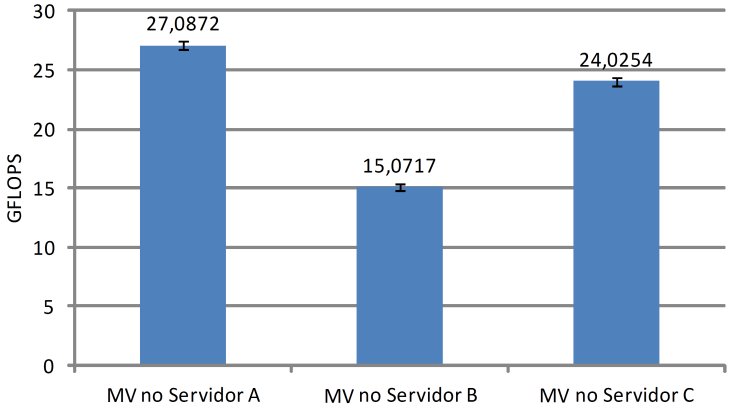


Figura 3.3: Experimento motivacional realizado nos Servidores A, B e C.

Este resultado é um exemplo do impacto que a diferença entre MFs causam no desempenho das aplicações em nuvens privadas. Entretanto, a variabilidade de desempenho é um problema que afeta tanto as nuvens privadas, quanto as nuvens públicas.

A imprevisibilidade do desempenho na nuvem é um grande problema para muitos usuários e é considerado como um dos maiores obstáculos para computação em nuvem (ARMBRUST et al., 2009). Por exemplo, os pesquisadores esperam um desempenho comparável para sua aplicações a qualquer momento, independente da carga de trabalho atual da nuvem ou de onde as MVs estão sendo alocadas, o que é muito importante para pesquisadores, por causa da repetibilidade de resultados.

Iosup, Yigitbasi e Epema (2011) afirmam que outro importante obstáculo para a adoção da computação em nuvem é confiar que os serviços em nuvem são, de fato, confiáveis. Como os recursos de computação em nuvem e *software* não são mais hospedados e operados pelo usuário, e os fornecedores não divulgam características de suas infraestruturas, nem como eles operam seus recursos físicos de forma compartilhada, fica difícil acreditar, por exemplo, que seu desempenho será estável durante longos períodos.

Diversos trabalhos citam o problema de variabilidade nos provedores de IaaS. Schad,

Dittrich e Quiané-Ruiz (2010) realizaram diversos experimentos com aplicações MapReduce na Amazon EC2, e perceberam que o tempo de execução pode variar consideravelmente, chegando a uma diferença de 24% em alguns casos. Iosup, Yigitbasi e Epema (2011) analisaram o desempenho de diversos serviços do Google e da Amazon, e detectaram uma grande variabilidade de desempenho, com vários exemplos de indicadores de performance que tiveram variação mensal média superior a 50%. Dejun, Pierre e Chi (2009) também estudaram a variação de desempenho na Amazon EC2, focando principalmente no desempenho das aplicações Tomcat e MySQL, que chegou a atingir uma variação de 71,1% para um tipo de suas operações.

Dado os fatos apresentados, acreditamos que seja importante, principalmente para nuvens privadas, a criação de uma estratégia para resolver o problema de prover recursos computacionais, especificamente poder de processamento, em ambientes heterogêneos, onde existe uma grande variabilidade entre os processadores das MFs da infraestrutura. Neste trabalho, é proposta uma solução que visa padronizar a representação da capacidade de processamento em termos de unidades de processamento.

Como as CPUs das MFs podem apresentar poder de processamento diferente e a maioria das políticas de escalonamento utiliza os dados brutos (geralmente quantidade de CPUs) para fazer o escalonamento, torna-se essencial também, a criação ou adaptação de políticas de escalonamento, baseadas nas UPs.

4 SOLUÇÃO PROPOSTA

Neste capítulo será apresentada a solução proposta para o problema apresentado no capítulo anterior. As unidades de processamento são apresentadas na Seção 4.1, as ideias sobre a limitação do uso da CPU das MVs são apresentadas na Seção 4.2, e a arquitetura FairCPU e detalhes de implementação são apresentados nas Seções 4.3 e 4.4, respectivamente.

4.1 Unidade de Processamento

A utilização das unidades de processamento é a ideia chave deste trabalho. A UP é uma abstração utilizada para representar o poder de processamento de MFs e MVs. Ela deve ter um valor constante e conhecido, em termos de GFLOPS, MIPs ou outra métrica, e substituirá o valor bruto da quantidade de CPUs, que é o parâmetro utilizado na maioria dos *middleware* e atuais provedores de IaaS na hora da alocar as MVs.

Com a UP representando o poder de processamento efetivo de MVs e MFs, é possível prover uma alocação uniforme em uma infraestrutura heterogênea, uma vez que as MFs devem ter seu poder de processamento calculado (em termos de GFLOPS, por exemplo) e depois convertido em função das UPs, e todas as MVs devem solicitar uma quantidade de UPs para serem instanciadas. A ideia é manter o poder de processamento das MVs, independente da MF em que ela seja alocada.

Para exemplificar, vamos apresentar a ideia de maneira simplista. Primeiro, vamos supor a existência de 2 MFs (M_1 e M_2), cada uma delas com 4 CPUs (uso da CPU pode variar de 0 a 400%), mas com processadores diferentes; e que a UP foi definida como 2 GFLOPS pelo administrador da nuvem. Ao iniciar uma MV com 4 VCPUs em cada uma dessas MFs e executar o *benchmark* Linpack, supõe-se que o resultado foi de 20 GFLOPS para a MV que estava alocada na MF M_1 e 16 GFLOPS para a MV que estava alocada na MF M_2. Assim, teríamos que M_1 tem um poder de processamento de 10 UPs (20 GFLOPS $\div$ 2 GFLOPS), enquanto M_2 tem 8 UPs (16 GFLOPS $\div$ 2 GFLOPS). Sabendo que as duas MFs tem 4 CPUs, podemos dizer que 1 UP em M_1 equivale a 40% (400% $\div$ 10) de um *core* da máquina M_1, e 1 UP na máquina M_2 equivale a 50% (400% $\div$ 8) de um *core* da máquina M_2. Conhecendo essa percentagem, é possível aplicar o limite de utilização da CPU para as MVs e assim garantir o valor do poder de processamento, independente de onde a MV seja instanciada. Caso a MV seja alocada em M_1, deve-se limitar o uso da CPU em 40%, caso ela seja alocada em M_2, deve-se limitar o uso da CPU em 50%. Dessa maneira, a MV terá o mesmo desempenho nas duas MFs, pois 1 UP em M_1 é equivalente a 1 UP em M_2, 2 GFLOPS.

O exemplo apresentado é simplista, pois não considera as tecnologias de processadores que podem alterar o desempenho da máquina de acordo com a carga de trabalho, como as tecnologias Intel Turbo Boost e AMD Turbo Core. Com isso, a relação entre a percentagem da CPU utilizada e o desempenho da máquina pode sofrer variações, sendo necessária uma es-

tratégia[1] mais elaborada para definir a UP. De toda forma, a ideia principal é que se a MV está numa MF mais potente, ela deve precisar de menos tempo de CPU do que uma MV que está numa MF mais fraca. Assim, com os limites definidos corretamente, pode-se utilizar as UPs para garantir uma menor variabilidade no desempenho de MVs em execução num ambiente heterogêneo. Esta é a ideia por trás da utilização das UPs.

4.2 Limitando o uso da CPU

Para que a ideia apresentada na seção anterior seja colocada em prática, é preciso que os hipervisores permitam definir a percentagem da CPU que será utilizada pelas MVs. Neste trabalho, são considerados os dois hipervisores mais utilizados de código aberto, Xen e KVM. Uma vez que o KVM realiza virtualização assistida por *hardware* e o Xen paravirtualização, será possível utilizar MFs com processadores com e sem tecnologia de virtualização (Intel VT e AMD-V), respectivamente, tornando o ambiente ainda mais heterogêneo.

O Xen possui a funcionalidade para limitar o uso da CPU de MVs implementada nativamente através do seu algoritmo de escalonamento de CPU, *Credit Scheduler* (CREDITSCHEDULER, 2012). Utilizando esse algoritmo, para cada MV é atribuído um peso (*weight*) e um limite (*cap*). Através destes parâmetros é possível definir prioridade de processos, limitar o uso da CPU e definir se o processo será com ou sem conservação de trabalho. No modo com conservação do trabalho, caso a MV esteja em execução sem concorrência, ela utilizará todos os recursos disponíveis e só estará limitada quando outra MV estiver ativa e houver concorrência pela CPU. No outro modo, mesmo que não haja concorrência pela CPU, a MV utilizará apenas o limite que lhe foi dado (CHERKASOVA; GUPTA; VAHDAT, 2007).

A Figura 4.1 apresenta dois exemplos de configuração para o *Credit Scheduler*. Na primeira (Figura 4.1a), a MV App01 é configurada para utilizar 70% da CPU, mesmo que haja mais recursos livres. Na segunda (Figura 4.1b), a MV App02 utilizará todos os recursos disponíveis até que outra MV inicie execução na mesma MF. Neste caso, o peso será considerado para definir a proporção com que os recursos serão distribuídos entre as MVs. Por exemplo, uma MV que tiver peso 512 terá acesso ao dobro do tempo de CPU de uma MV com peso 256.

```
xm sched-credit -d App01 -c 70
```

(a) Configuração baseada no limite (*cap*: -c)

```
xm sched-credit -d App02 -w 256 -c 0
```

(b) Configuração baseada no peso (*weight*: -w)

Figura 4.1: Exemplos de configuração do *Credit Scheduler*.

O KVM, que foi lançado oficialmente em 2007, ainda não possui uma solução nativa que permita limitar o uso da CPU das MVs. Apesar desse fato, as MVs criadas com KVM são processos do Linux e integram-se perfeitamente com o restante do sistema. Aproveitando disso, foi avaliada a utilização da ferramenta *cpulimit* (CPULIMIT, 2012) para definir a percentagem da CPU que vai ser utilizada pela MV.

[1] A estratégia utilizada para definir a UP de todos os experimentos desse trabalho é apresentada na Seção 5.1.1.

4.2.1 *Cpulimit*

A ferramenta *cpulimit*, cuja última versão estável é a 1.1, é um programa de código aberto utilizado para limitar o uso da CPU de um processo no Linux. Ele age no uso da CPU real e é capaz de se adaptar à carga total do sistema de forma dinâmica e rápida.

O trabalho do *cpulimit* é realizado inteiramente no espaço do usuário, de modo que ele não interfere no escalonador do Linux. O processo é continuamente interrompido e reiniciado, enviando-lhe sinais *SIGSTOP*[2] e *SIGCONT*[3]. Os sinais são enviados pelo *cpulimit* nos momentos adequados, com base no limite especificado pelo usuário e as estatísticas do processo, que são lidas a partir de */proc*.

A Figura 4.2 apresenta um exemplo de execução da ferramenta *cpulimit*, onde a MV, cujo identificador de processo é igual a 4432, é limitada em 75% do uso de CPU. A ferramenta, que neste exemplo foi executada em modo verboso, apresenta a quantidade de tempo que o processo ficou ativo (*work quantum*) e dormindo (*sleep quantum*).

```
root@host01:~# cpulimit -p 4432 -l 75 -v
Process 4432 detected

%CPU    work quantum    sleep quantum    active rate
76.38%      72934 us        27065 us        74.28%
75.26%      74347 us        25652 us        74.61%
75.25%      74467 us        25532 us        74.72%
```

Figura 4.2: Exemplo de execução da ferramenta *cpulimit* no modo verboso.

Para descobrir se o *cpulimit* poderia ser utilizado para limitar o uso da CPU de MVs com o hipervisor KVM, foram realizados diversos experimentos (REGO; COUTINHO; SOUZA, 2011). Além disso, foram necessárias algumas alterações no código-fonte[4] da ferramenta para que ela trabalhasse corretamente com processos que utilizam mais de um *core* da CPU (MVs com duas ou mais VCPUs), para que fosse possível aplicar limites de 150%, 175% ou 220%, por exemplo.

4.3 Arquitetura FairCPU

A arquitetura FairCPU tem o objetivo de tornar homogênea a alocação de poder computacional às MVs e padronizar a representação do poder de processamento de uma infraestrutura de nuvem. As ideias sobre a utilização das UPs e a limitação do uso da CPU, discutidas nas seções anteriores, são utilizadas pela arquitetura para prover poder computacional de forma homogênea, mesmo estando em um ambiente heterogêneo, composto por MFs com diferentes configurações e utilizando diferentes hipervisores e técnicas de virtualização.

[2]Nome do sinal emitido pelo sistema operacional, que faz com que um processo pare a sua execução enquanto não receber o sinal *SIGCONT*.

[3]Nome do sinal enviado para reiniciar um processo pausado por um sinal *SIGSTOP* ou *SIGTSTP*.

[4]O código-fonte com as alterações realizadas será disponibilizado para a comunidade, seguindo a licença da ferramenta, GPLv2.

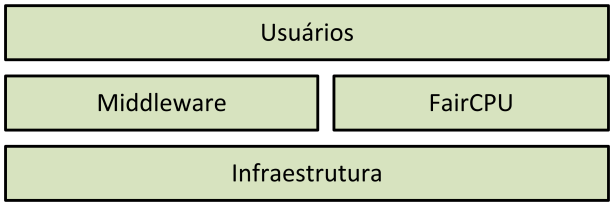


Figura 4.3: A arquitetura FairCPU trabalha na camada de IaaS.

A FairCPU trabalha na mesma camada dos *middleware* de computação em nuvem (Figura 4.3), mas não os substitui. Ela estende as funcionalidades do *middleware* para trabalhar com as UPs, e assim modificar a forma como o poder de processamento é representado e alocado para as MVs. Seis módulos compõem a arquitetura: Gerenciador de Limites, Monitor, Escalonador, *Logger*, *Daemon* e Cliente. As relações entre os componentes da arquitetura pode ser vistas na Figura 4.4.

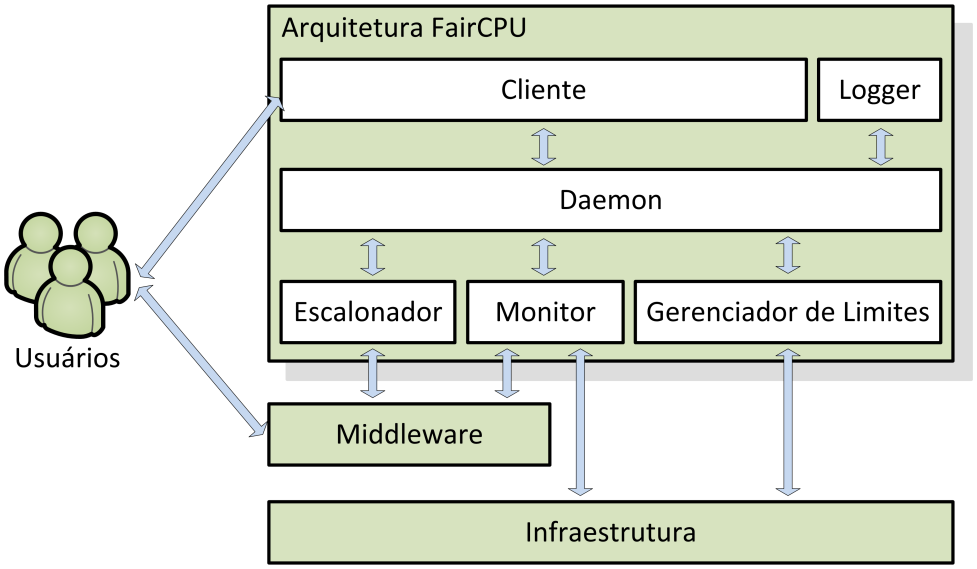


Figura 4.4: Visão geral da arquitetura proposta.

Gerenciador de Limites: é o módulo responsável pela aplicação dos limites de uso de CPU. Quando uma nova MV é instanciada, por exemplo, o *Daemon* invoca o Gerenciador de Limites, que configura quanto da CPU a MV pode utilizar baseado na quantidade de UPs requisitadas para ela. O Gerenciador de Limite também é invocado para ajustar o poder de computação das MVs após migrações ou quando a quantidade de UPs é modificada (ver módulo Cliente). Este módulo é o único que, obrigatoriamente, necessita de comunicação direta com as MFs da infraestrutura para que os limites de uso da CPU sejam aplicados.

Monitor: é o modulo responsável pela captura de informações sobre a infraestrutura da nuvem. O módulo pode acessar diretamente os dados ou fazer requisições ao *middleware* de computação em nuvem. As informações monitoradas referem-se à quantidade de CPU, memória, armazenamento e rede de cada MF e MVs da infraestrutura. Estas informações são utilizadas pelos demais módulos da arquitetura.

Escalonador: é o módulo responsável por decidir em qual MF as MVs devem ser alocadas. Diferente dos escalonadores presentes nos principais *middleware* de computação em nuvem, como Eucalyptus e OpenNebula, o escalonador da arquitetura FairCPU não deve considerar as informações brutas sobre CPU, memória e disco para definir onde as MVs serão alocadas. Uma vez que a arquitetura é baseada em UPs, todas as alocações devem ser feitas baseadas na quantidade de UPs requisitadas e na quantidade de UPs livres nas MFs.

Logger: é o módulo responsável por catalogar todas as operações e mensagens da arquitetura. Sua principal função é facilitar o trabalho do administrador da infraestrutura, ao ajudar no processo de depuração e detecção de problemas.

Daemon: é responsável por gerenciar todos os módulos e, assim, controlar a infraestrutura. Ele está em contato direto com o *middleware* de computação em nuvem e mantêm informações sobre a MVs e MFs em execução (capturadas pelo módulo Monitor), além de ter acesso a todas as requisições que chegam ao *middleware*. O *Daemon* tem a responsabilidade de invocar o módulo de Escalonamento quando houverem MVs no estado pendente e alocar as MVs, assim que o Escalonador retornar o mapeamento MV-MF. Ele é responsável também por armazenar informações sobre as UPs e invocar o módulo de Gerenciamento de Limites para aplicar o limite no uso da CPU sempre que uma MV for iniciada ou migrada.

Cliente: é o módulo que permite que os usuários interajam com a arquitetura. Através dele, os usuários podem requisitar informações sobre as MFs e MVs (incluindo a quantidade de UPs disponíveis), além de poder alterar a quantidade de UPs de uma MV em execução. Este módulo encaminha as requisições dos usuários ao *Daemon*, que retorna com as informações requisitadas.

A arquitetura FairCPU pode trabalhar de maneira transparente ao usuário, caso o módulo Cliente seja desabilitado, pois ela foi projetada para não interferir nas interfaces de acesso do *middleware*. Com seu uso, as MVs solicitadas pelo usuário terão poder computacional equivalente, independentemente da MF subjacente. Um exemplo de uso é a requisição de um *cluster* virtual, onde cada nodo é alocado em uma MF diferente, e, consequentemente, podem possuir poder de processamento distintos. Com o uso da alocação baseada em UPs, todos os nodos possuem capacidades equivalentes. Essa característica é importante para o balanceamento de carga e sincronização das aplicações, bem como para migração de MVs, garantindo desempenhos semelhantes independente de para onde a VM seja migrada. Outra característica interessante é a possibilidade de aumentar e diminuir dinamicamente a quantidade de UPs de uma MV, o que pode ser utilizado para ajustar rapidamente os recursos para diferentes cargas de trabalho.

A Figura 4.5 mostra como a arquitetura funciona, uma vez que apresenta o comportamento do *Daemon* e as principais decisões tomadas por ele, bem como alguns dos subprocessos executados pelos módulos Escalonador, Gerenciador de Limites e Cliente. Em seguida, são apresentados os três casos de uso mais comuns da arquitetura.

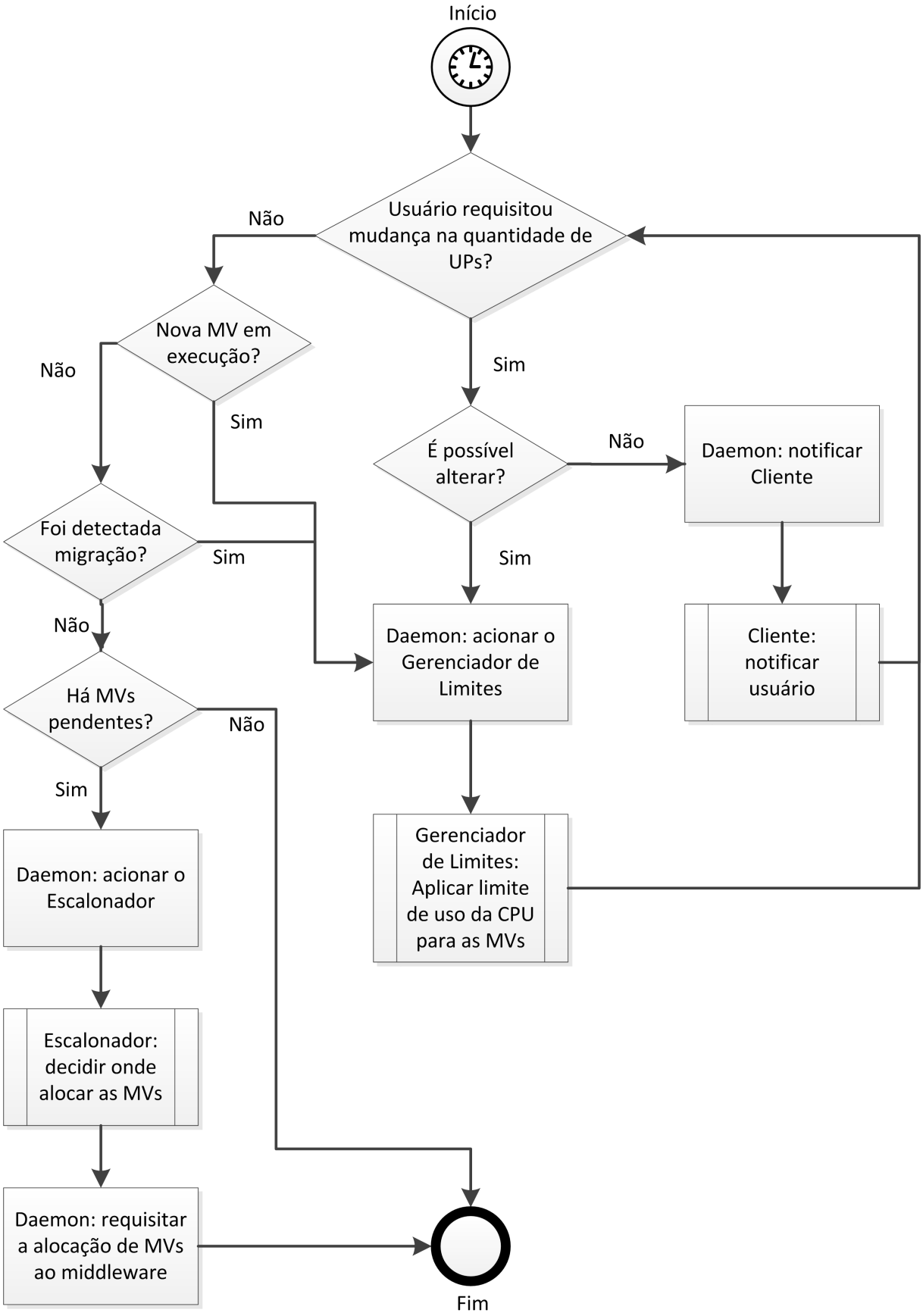


Figura 4.5: Fluxograma apresentando o comportamento da arquitetura FairCPU.

Caso 1 - Instanciação de duas MVs:

1. De tempos em tempos, o *Daemon* chama o Monitor para capturar informações da infra-estrutura;
2. Usuário requisita ao *middleware* a criação de 2 MVs (MV_A e MV_B);
3. *Daemon* detecta a existência de MVs no estado pendente;
4. *Daemon* chama o Escalonador;
5. Escalonador retorna o mapeamento MV-MF para o *Daemon*;
6. *Daemon* requisita ao *middleware* a alocação das MVs;
7. *Daemon* detecta que a MV_A entrou em execução e chama o Gerenciador de Limites;
8. Gerenciador de Limites aplica o limite de uso da CPU na MV_A;
9. *Daemon* detecta que a MV_B entrou em execução e chama o Gerenciador de Limites;
10. Gerenciador de Limites aplica o limite de uso da CPU na MV_B.

Caso 2 - Migração de MV:

1. Usuário requisita ao *middleware* a migração da MV_A da MF_1 para a MF_2;
2. *Daemon* detecta a migração da MV;
3. *Daemon* chama o Gerenciador de Limites;
4. Gerenciador de Limites aplica o limite de uso da CPU na MV_A.

Caso 3 - Alteração na quantidade de UPs da MV:

1. Usuário solicita ao módulo Cliente a alteração na quantidade de UPs da MV_A;
2. *Daemon* recebe a requisição e, se for possível aumentar, chama o Gerenciador de Limites, senão retorna mensagem avisando que a MF, em que a MV está, não tem UPs livres;
3. Gerenciador de Limites aplica o novo limite de uso da CPU para a MV_A.

4.4 Implementação

A arquitetura FairCPU pode ser implementada e executada em conjunto com diversos *middleware*. Como estudo de caso, foi escolhido o *middleware* OpenNebula baseado nas características e funcionalidades oferecidas. A principal delas é a flexibilidade para a criação das MVs, pois é possível configurar os recursos de acordo com as necessidades das aplicações, diferente do Eucalyptus e OpenStack, onde é preciso escolher uma classe pré-configurada. Além disso, o prévio conhecimento da ferramenta, que está sendo utilizada na infraestrutura do nosso laboratório de pesquisa, também foi um fator considerado para a escolha.

FairCPU foi implementada sobre o *middleware* OpenNebula versão 2.2.1, utilizando a linguagem Ruby versão 1.8 e *OpenNebula Cloud API* (OCA) 2.2, além de alguns *scripts shell*. A única alteração feita no *middleware* OpenNebula foi desabilitar o escalonador padrão, o que pode ser feito diretamente na configuração do OpenNebula.

A definição da quantidade de UPs que será alocada para a MV também não precisou alterar o *middleware*. No OpenNebula, a configuração das MVs é feita através de *templates*, onde define-se a imagem a ser utilizada, bem como a quantidade de VCPUs, memória e outros parâmetros. O parâmetro "UP"é utilizado para definir quantas UPs serão alocadas para a MV. O valor que for definido pelo usuário para este parâmetro será utilizado pela arquitetura para definir o poder de processamento alocado para a MV. A Figura 4.6 apresenta um exemplo de *template*, em que é requisitada a criação de uma MV com 2 VCPUs, 2 GB de memória e 4 UPs. Quando o módulo Cliente estiver desabilitado ou esse parâmetro não for definido no *template*, ele será considerado igual ao número de VCPUs.

```
NAME     = "MV_exemplo2"
VCPU     = 2
UP       = 4

MEMORY = 2048

DISK   = [ image = "DebianSqueeze" ]

DISK   = [ type       = swap,
           size       = 1024,
           readonly = "no" ]

NIC    = [ network = "RedePrivada" ]
```

Figura 4.6: Exemplo de *template* para criação de MV com poder de processamento de 4 UPs.

A Figura 4.7 mostra uma infraestrutura típica de nuvem privada, onde o *middleware* é executado em uma MF chamada de controlador (ou *front-end*), enquanto as outras MFs, chamadas de nodos, são responsáveis por executar as MVs. A implementação da arquitetura FairCPU é executada também no controlador e se comunica com as outras MFs utilizando SSH (*Secure shell*).

Os usuários geralmente acessam o *middleware* através de um portal, que é comumente hospedado no controlador, ou através de linha de comando no *shell* do controlador. Já

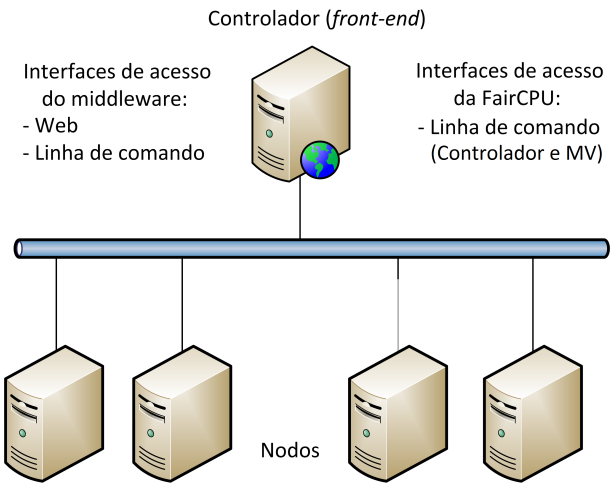


Figura 4.7: Organização típica de nuvem privada, composta de um controlador e diversos nodos. A FairCPU é executada no controlador e é através dele que os usuários acessam as funcionalidades da arquitetura.

o acesso às funcionalidades da FairCPU tem que ser feito através do *shell*, porém, é possível executar o módulo Cliente tanto no *shell* do controlador (assim como o *middleware* é operado), quanto no *shell* da MV.

As próximas seções apresentam mais detalhes sobre como alguns dos módulos e funcionalidades foram implementados.

4.4.1 Gerenciador de Limites

Como discutido na Seção 4.2, o Xen tem uma funcionalidade nativa para limitar o uso da CPU através do algoritmo *Credit Scheduler*, enquanto o KVM não tem, por isso foi utilizada a ferramenta *cpulimit* para esse propósito. Dessa forma, a arquitetura FairCPU pode trabalhar com os dois hipervisores, o que torna este trabalho mais complexo, por habilitar sua utilização em ambientes bastante heterogêneas a nível de *hardware* e *software*.

O Gerenciador de Limites tem acesso à todas as informações das MVs, pois sabe em que MF ela foi alocada e qual hipervisor está em execução nesta MF. Dessa forma, o Gerenciador de Limites, através de *scripts shell* e SSH, pode executar a ferramenta correta para aplicar o limite no uso da CPU, seja o *cpulimit* para KVM ou o *Credit Scheduler* para Xen.

Para que a arquitetura garanta que a UP terá o mesmo poder computacional em todas as MFs, é preciso definir o poder de processamento de uma UP e qual o limite que precisa ser aplicado à CPU para atingir esse poder computacional. Para isso, é preciso descobrir o poder de processamento das MVs em cada MF, bem como seu comportamento quando ocorre o aumento da carga de trabalho.

Hoje, existem diversas tecnologias que interferem diretamente no desempenho dos processadores (e consequentemente, no desempenho das MVs), como regulagem dinâmica de voltagem e frequência (*dynamic voltage and frequency scaling* - DVFS), AMD Turbo Core, In-

tel Turbo Boost e Intel Hyper-Threading. Estas tecnologias tornam muito complexo o processo de definir a relação "limite-poder de processamento". Neste trabalho, esta relação para cada MF foi descoberta de forma não automatizada, ao executar o *benchmark* Linpack em varias MVs, com diferentes limites de uso da CPU, e comparar os resultados. Mais detalhes podem ser encontrados em (REGO; COUTINHO; SOUZA, 2011; REGO et al., 2011) e no próximo capítulo. Implementar um programa que defina a relação "limite-poder de processamento"de forma automatizada para diferentes configurações de MF é um trabalho futuro bastante interessante e complexo.

4.4.2 Cliente

Cargas de trabalho dinâmicas são muito comuns em ambientes de computação em nuvem. Por isso, o módulo Cliente fornece a funcionalidade de alterar dinamicamente a quantidade de UPs alocadas para uma MV. Esta é uma forma de prover escalabilidade vertical sem precisar desligar e reiniciar a MV, não interferindo na execução de aplicações e serviços.

Como a arquitetura trabalha com limites no uso da CPU para prover as UPs, a estratégia adotada foi criar todas as MVs com o máximo de VCPUs possíveis e deixar todo o gerenciamento relacionado ao poder computacional com o Gerenciador de Limites.

A utilização dessa estratégia é segura, pois todo o controle é feito a nível de hipervisor pelo Gerenciador de Limites. De nada adianta a MV ter 2, 4 ou 8 VCPUs se ela está configurada para usar apenas 1 UP. Dessa forma, qualquer alteração na quantidade de poder computacional, seja para aumentar ou diminuir a quantidade de UPs, pode ser feita em milissegundos, provendo escalabilidade vertical e se adaptando rapidamente às mudanças de carga de trabalho[5].

O módulo Cliente se comunica com o *Daemon* seguindo o paradigma cliente-servidor, dessa forma, é possível executá-lo de dentro das MVs, permitindo que o usuário solicite a mudança da quantidade de UPs rapidamente por linha de comando, ou crie um *script* ou programa que agende essa mudança. A Tabela 4.1 apresenta a lista dos comandos que podem ser utilizados pelos usuários.

Tabela 4.1: Comandos que podem ser utilizados através do módulo Cliente.

Comando	**Descrição**
list vms	Lista as MVs
list hosts	Lista as MFs da nuvem
setpu <ID_{MV}> <$quantidade_{UPs}$>	Altera a quantidade de UPs alocadas para a MV cujo identificador é <ID_{MV}>

Além da possibilidade de alterar dinamicamente a quantidade de UPs alocadas à MV, é possível listar as MFs e MVs da infraestrutura, e descobrir quantas UPs estão livres nas MF e quantas estão sendo usadas pelas MVs. Com esse módulo, a mudança na representação da capacidade de processamento da nuvem fica visível para o usuário.

[5] A arquitetura não se adapta sozinha às mudanças de carga de trabalho. Ela apenas fornece as funcionalidades para isso.

Um detalhe importante é que pode não ser possível aumentar a quantidade de UPs da MV, por não haver UPs livres na MF. Dessa forma, o usuário precisa solicitar a migração da MV ou liberar UPs em alguma outra MV, que esteja em execução na mesma MF. Implementar uma funcionalidade para automatizar esse processo é um trabalho futuro interessante.

4.4.3 Escalonador

O OpenNebula vem, por padrão, com o escalonador *Match-making* que implementa políticas de escalonamento baseada em *Rank* (OPENNEBULA, 2012). Este escalonador pode utilizar três políticas de alocação: *Packing Policy*, *Striping Policy* e *Load-aware Policy*. Estas políticas são heurísticas que visam minimizar o número de MFs em uso ou maximizar os recursos disponíveis em uma MF, escolhendo o servidor com mais MVs em execução, com menos MVs em execução ou com mais CPU livre, respectivamente.

Como a arquitetura FairCPU é baseada em UPs, essas mesmas políticas de escalonamentos foram reimplementadas, só que agora elas são baseadas nos números de UPs requisitadas pelas MVs e disponíveis nas MFs, e não mais no número bruto de CPUs. A Tabela 4.2 mostra o objetivo e heurística de cada uma das políticas padrões do OpenNebula.

Tabela 4.2: Objetivos e heurísticas das políticas de escalonamento do OpenNebula.

	Packing Policy
Objetivo	Minimizar o número de MFs em uso
Implementação	Alocar a MV no nodo com mais MVs em execução
	Striping Policy
Objetivo	Espalhar as MVs entre os nodos
Implementação	Alocar a MV na MF com menos MVs em execução
	Load-aware Policy
Objetivo	Maximizar os recursos disponíveis para as MVs em um nodo
Implementação	Alocar a MV na MF com mais UPs livres

Além das políticas já apresentadas, uma política aleatória e uma heurística para minimizar o gasto de energia do *datacenter* (*Energy-aware)* foi adaptada e melhorada de (YOUNGE et al., 2010). Para isso, um wattímetro foi utilizado para medir o consumo de energia por UP em cada MF, como ilustra a Figura 4.8.

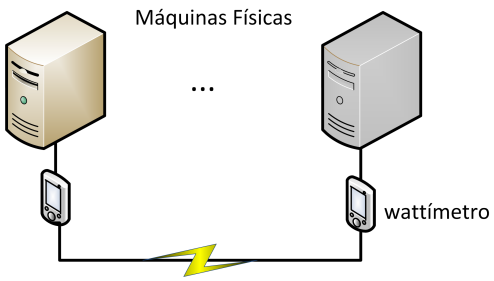


Figura 4.8: Cada MF foi ligada a um wattímetro, e este ligado à corrente elétrica. Dessa maneira, foi possível medir o consumo de energia por UP, em cada MF da infraestrutura.

Como o consumo de energia está diretamente relacionado ao uso e tipo de CPU

(KRISHNAN et al., 2011), foi possível criar uma heurística que escolhe a MF que consome menos energia para executar a MV.

O Algoritmo 1 ordena a lista de MVs em ordem decrescente de quantidade de UPs requisitadas e, através do Algoritmo 2, seleciona a MF que gastará menos energia para executar cada MV. Essa política escolhe onde alocar as MVs considerando sempre o caso em que todas as MVs estarão utilizando toda a CPU disponível. A ideia de ordenar a lista de MVs e considerar MFs heterogêneas são algumas das alterações feitas no algoritmo original.

A relação de MFs e o consumo de energia por UP em cada uma delas são apresentados no próximo capítulo, que é destinado aos experimentos realizados para auxiliar no processo de avaliação da arquitetura.

Algoritmo 1: *Energy-aware*. Algoritmo de alocação de MVs com objetivo de reduzir consumo de energia do *datacenter*, ao alocar as MVs nas MFs que consomem menos energia.

```
1 Ordena lista_MVs por número decrescente de UPs;
2 for i = 1 → |queue| do
3     mv = queue[i];
4     indice = SelMin(lista_MFs, mv.UPs, mv.memoria);
5     if indice > 0 then
6         mf = lista_MFs[indice];
7         Aloca mv em mf;
8     else
9         Não é possível alocar mv;
```

Algoritmo 2: SelMin. Algoritmo que escolhe a MF, que gasta menos energia para executar uma MV, baseado no estado atual da infraestrutura ($lista_{MFs}$) e na quantidade de UPs e memória requisitas (mv_{UPs} e $mv_{memoria}$).

```
Entrada: lista_MFs, lista_MFs, mv_memoria
Saída: Índice da MF em que ocorrerá o menor gasto de energia
1  min = 1000000;
2  escolha = 0;
3  for j = 1 → |lista_MFs| do
4      mf = lista_MFs[j];
5      if lista_MFs ≤ mf.UPs then
6          if mv_memoria ≤ mf.memoria then
7              gasto = gasto em watts para mv.UPs em mf;
8              if gasto < min then
9                  min = gasto;
10                 escolha = j;
11     j++;
12 return escolha
```

5 EXPERIMENTAÇÃO E DISCUSSÃO DOS RESULTADOS

O presente capítulo destina-se à apresentação dos experimentos computacionais conduzidos com o propósito de avaliar a arquitetura FairCPU.

5.1 Metodologia

Todos os experimentos foram realizados numa nuvem privada heterogênea, composta por 3 máquinas do tipo Ci5 (uma delas sendo o controlador), 1 máquina do tipo Ci7, 1 máquina do tipo X4, e 1 máquina do tipo C2D, todas conectadas a uma rede Gigabit. As configurações das MFs podem ser vistas na Tabela 5.1. Em todo o capítulo, a MFs serão identificadas pelo tipo.

Tabela 5.1: Configuração das Máquinas Físicas.

Tipo	Processador	UPs	Memória	Sistema Operacional	Hipervisor
Ci7	Intel Core i7-930 @ 2,80 GHz	10	24 GB	Ubuntu Server 11.04 64bits	KVM
Ci5	Intel Core i5-750 @ 2,66 GHz	9	4 GB		
X4	Intel Xeon X3430 @ 2,40 GHz	8	8 GB		
C2D	Intel Core 2 Duo E7400 @ 2,80 GHz	5	4 GB	Debian 6.0 64 bits	Xen

A nuvem privada foi criada com o *middleware* OpenNebula versão 2.2.1, configurado para utilizar o sistema de arquivos compartilhado NFS (*Network File System*) para distribuição das MVs. A arquitetura FairCPU, cujos detalhes de implementação foram apresentados na Seção 4.4, foi incorporada ao sistema.

As seguintes aplicações foram utilizadas nos experimentos:

Intel Linpack: é uma generalização do *benchmark* Linpack 1000. Ele resolve um sistema denso de equações lineares, e mede a quantidade de tempo que leva para fatorar e resolver o sistema. Após isso, ele converte o tempo em uma taxa de desempenho e testa os resultados por precisão. A generalização está no número de equações que este *benchmark* pode resolver, que não se limita a 1000, diferente da versão original (INTEL, 2012).

Problema de transferência de calor (*heat problem*): problema clássico de aplicações de métodos numéricos em fenômenos físicos. A transferência de calor em uma placa plana homogênea é analisada, considerando que todos os pontos internos da placa estejam a uma temperatura inicial diferente das temperaturas das bordas (GALANTE; BONA, 2011). O problema consiste em determinar a temperatura em qualquer ponto interno da placa em um dado instante de tempo. A aplicação foi implementada com a linguagem C e OpenMP (DORTA; RODRIGUEZ; SANDE, 2005).

Algoritmo de redução LU (*LU reduction*): é um algoritmo relacionado à decomposição LU, sendo uma versão paralelizada deste. Costuma ser utilizado como *benchmark* no contexto de computação paralela. A aplicação foi implementada com a linguagem C e OpenMP (DORTA; RODRIGUEZ; SANDE, 2005).

Problema de multiplicação de matrizes: é uma operação fundamental em muitas aplicações numéricas de álgebra linear. Sua implementação eficiente em computadores paralelos é uma questão de primordial importância para bibliotecas de *software* científico. A aplicação foi implementado em C, utilizando a biblioteca OpenMPI para a troca de mensagens.

IOzone: é um *benchmark*, implementado em C, para sistema de arquivos. Ele realiza e mede diversas operações de arquivo, como escrita e leitura aleatória (IOZONE, 2012).

Iperf: é um *benchmark* para medir a largura de banda de redes de computadores. É um programa de código aberto, implementado em C++, e pode ser utilizado com fluxos de dados TCP e UDP (TIRUMALA; COTTRELL; DUNIGAN, 2003).

Para todos os experimentos, os *benchmarks* Linpack, IOzone e Iperf foram executados 30 vezes para cada configuração de MV e MF, enquanto as aplicações de transferência de calor e redução LU foram executadas 10 vezes. Em todos os casos, os resultados foram combinados para calcular a média, os limites inferior e superior com 95% de confiança. Já a aplicação de multiplicação de matrizes foi executada apenas uma vez.

5.1.1 Definição das UPs

A definição da UP para cada MF é um processo importante para garantir a eficiência da arquitetura. Por isso, a estratégia que foi utilizada para definir a UP é apresentada a seguir:

1. Descobrir o poder de processamento de cada MF. Para isso, uma MV com o máximo de VCPUs é criada em cada MF. O *benchmark* Linpack é executado nas MVs, afim de descobrir o poder de processamento de cada MF, já contando com a sobrecarga causada pela virtualização.

2. Escolher o valor da UP. Uma vez descoberto o poder de processamento das MFs, o administrador da nuvem precisa escolher um valor para representar a UP. Neste trabalho, foi definido que a UP seria equivalente a 3 GFLOPS, ou 50% de um *core* de uma máquina do tipo X4.

3. Descobrir a percentagem da CPU que equivale à UP, para cada MF. Para isso, é preciso executar o Linpack e aplicar diferentes limites no uso da CPU das MVs, até atingir o valor da UP. A Tabela 5.2 apresenta o limite que precisa ser aplicado para que as MVs tenham poder de processamento equivalente, independente da MF subjacente.

4. Definir quantas UPs cada MF terá. Baseado no poder computacional da MF e no valor da UP, é possível definir quantas UPs cada MF terá. Os valores já foram apresentados na Tabela 5.1.

O foco deste trabalho são as aplicações científicas, que exigem muito poder computacional, por isso a UP é definida com auxílio do *benchmark* Linpack e é baseada na quantidade

Tabela 5.2: Relação entre a percentagem da CPU e o poder de processamento das MFs. Valores acima de 400% para Ci7, Ci5 e X4 e 200% para C2D não se aplicam (NA), pois elas tem 4 e 2 CPUs, respectivamente.

	1 UP	**2 UPs**	**3 UPs**	**4 UPs**	**5 UPs**	**6 UPs**	**7 UPs**	**8 UPs**	**9 UPs**	**10 UPs**
Ci7	39%	78%	119%	157%	196%	234%	274%	320%	368%	400%
Ci5	41%	81%	118%	162%	218%	260%	305%	354%	400%	NA
X4	50%	100%	150%	200%	250%	300%	350%	400%	NA	NA
C2D	44%	83%	130%	165%	200%	NA	NA	NA	NA	NA

de GFLOPS. É importante salientar que a definição da UP é uma tarefa do administrador da nuvem, e que diversos outros *benchmarks*, métricas e estratégias podem ser utilizados para definir o valor das UPs e adequar a nuvem para um certo tipo de aplicação. A maneira como essa UP é definida impacta diretamente na variação de desempenho das MVs na nuvem, e pode ser um diferencial competitivo. Por exemplo, a Amazon EC2 não revela qual a estratégia utilizada para definir a sua unidade computacional, ECU (*Elastic Compute Unit*)[1].

5.2 Experimentos Computacionais

5.2.1 Experimento 1 - Análise das técnicas para limitar o uso da CPU

Este experimento visa avaliar a utilização da ferramenta *cpulimit* para limitar o uso da CPU das MV. Como a *cpulimit* não é uma solução nativa do hipervisor KVM, estes testes são importantes para validar sua eficiência. Para isto, o Linpack foi executado 30 vezes em uma MV (1 VCPU, 1GB de memória, SO Ubuntu Server 11.04 64 bits), sujeita a limites variados de utilização de CPU.

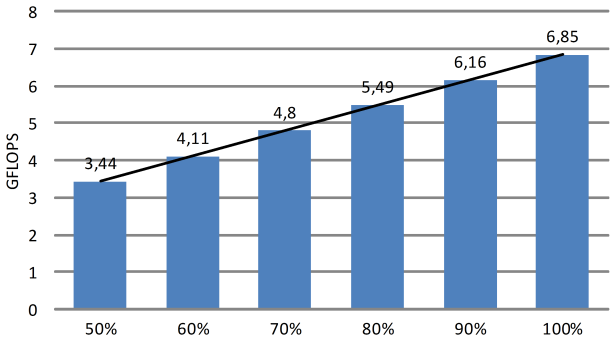


Figura 5.1: Variação do poder de processamento para a MF Ci5.

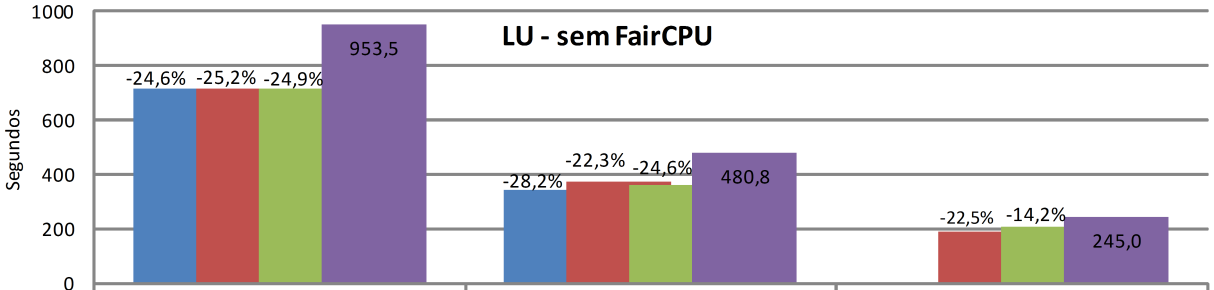
A Figura 5.1 mostra a variação do poder de processamento, ao variar a porcentagem da CPU alocada para a MV (em execução em uma MF do tipo Ci5). Foram utilizados limites de 50%, 60%, 70%, 80%, 90% e 100% para analisar a tendência de crescimento dos resultados, onde é possível notar uma curva de crescimento quase linear e proporcional ao valor de limitação. Estes resultados reforçam a escolha da ferramenta *cpulimit*, uma vez que constata-se que a MV obtém um desempenho proporcional ao limite aplicado.

[1] Uma ECU fornece a capacidade de CPU equivalente a um processador 2007 Opteron ou 2007 Xeon de 1,0-1,2 GHz. Isso também é o equivalente a um processador anterior a 2006 Xeon de 1,7 GHz (AMAZON, 2012b).

5.2.2 Experimento 2 - Eficiência da arquitetura FairCPU

O objetivo deste experimento é avaliar o impacto da diferença de poder de processamento das MFs no desempenho das aplicações em execução nas MVs. Duas aplicações OpenMP, uma de transferência de calor e a outra de redução LU, foram executadas em MVs com três diferentes configurações (1, 2 e 4 VCPUs, 2 GB de memória, Ubuntu Server 11.04 32 bits), sobre diferentes MFs.

Na primeira etapa deste experimento, a arquitetura FairCPU não estava em execução. As Figuras 5.2 e 5.3 mostram[2] o tempo de execução da aplicação de redução LU e transferência de calor, respectivamente, para cada configuração e tipo de MF[3]. Pode ser visto que ambas as aplicações demoraram mais tempo para executar na MF do tipo X4, para todas as configurações. A diferença entre os tempos de execução chega a 30,3%, no caso da aplicação de transferência de calor, com 1 VCPU, e 28,2% para o problema de redução LU, com 2 VCPUs. Este resultado reforça a influência da MF subjacente no desempenho das MVs e nas aplicações encapsuladas nelas.



	1 VCPU	2 VCPUs	4 VCPUs
C2D	718,2	344,9	
Ci7	713,1	373,5	189,7
Ci5	715,5	362,5	210,0
X4	953,5	480,8	245,0

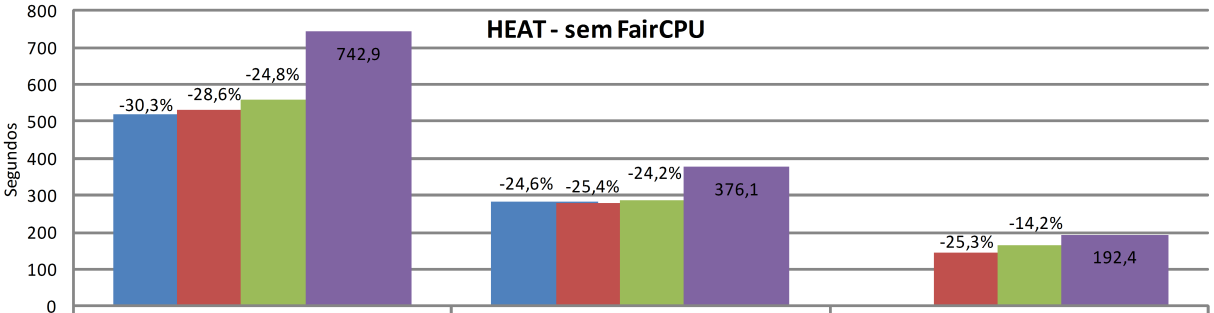
Figura 5.2: Tempo de execução da aplicação de redução LU, com a arquitetura FairCPU desabilitada.

A segunda etapa deste experimento foi realizada com a arquitetura FairCPU em execução. As mesmas aplicações foram executadas em MVs com 1, 2 e 4 VCPUs. Os resultados podem ser vistos nas Figuras 5.4 e 5.5. Como esperado, o tempo de execução é maior do que no caso anterior, pois a FairCPU, através do módulo Gerenciador de Limites, aplica limites no uso da CPU, não deixando a MV ter acesso a 100% da CPU das MFs. Neste caso, as MVs estavam configuradas para usar 1 UP por VCPU, ficando com 1, 2 e 4 UPs.

Tomando a máquina X4 como base, a maior diferença entre os tempos de execução do algoritmo de redução LU foi de 1,7% (usando 1 VCPU), enquanto o problema de transferência de calor teve uma variação máxima de 6% no tempo de execução, também para a configuração com 1 VCPU.

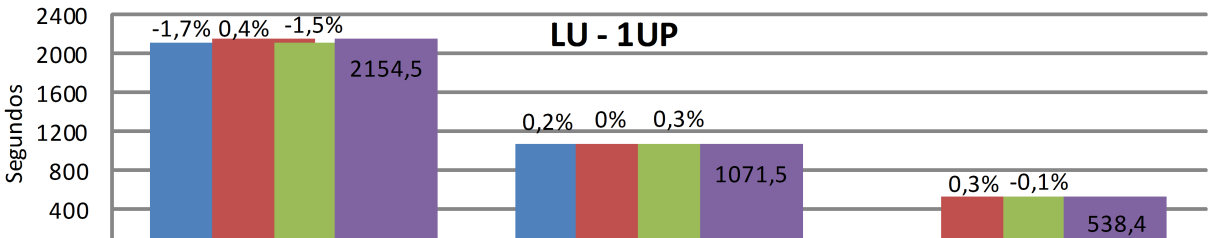
[2]A percentagem, apresentada em todas as figuras desta seção, é a variação do tempo de execução de cada experimento com relação ao tempo de execução na máquina do tipo X4.

[3]Como o processador da MF C2D tem apenas 2 núcleos, os testes com 4 VCPUs não foram realizados nessa máquina.



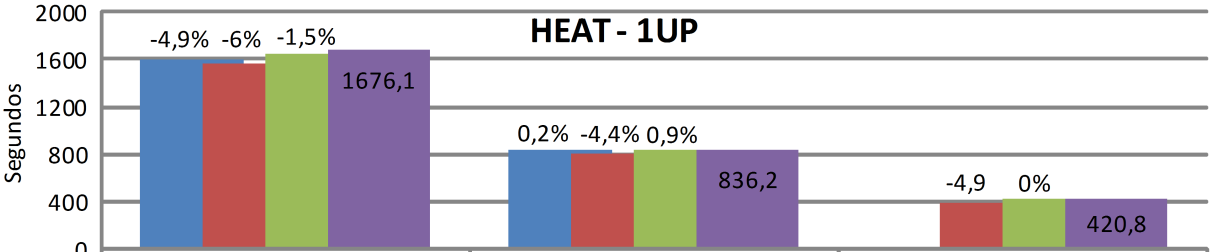
	1 VCPU	2 VCPUs	4 VCPUs
C2D	517,4	283,3	
Ci7	530,4	280,3	143,6
Ci5	558,1	284,9	164,9
X4	742,9	376,1	192,4

Figura 5.3: Tempo de execução da aplicação de transferência de calor, com a arquitetura FairCPU desabilitada. A percentagem apresentada é a variação do tempo do tempo de execução de cada experimento com relação ao tempo de execução na máquina do tipo X4.



	1 VCPU	2 VCPUs	4 VCPUs
C2D	2116,1	1069,0	
Ci7	2163,4	1071,0	540,2
Ci5	2120,4	1074,9	537,7
X4	2154,5	1071,5	538,4

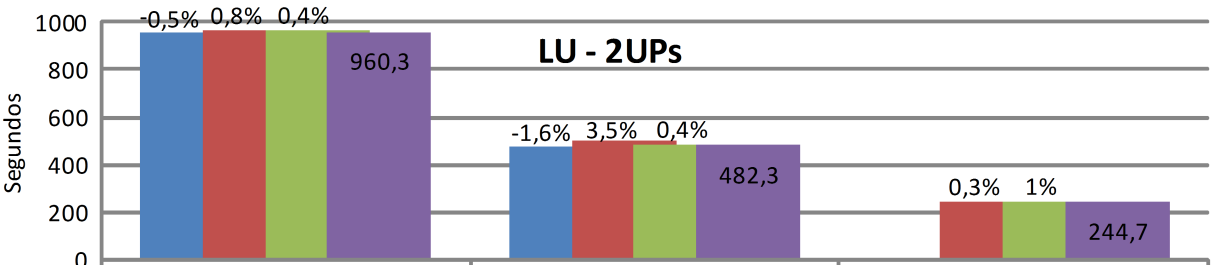
Figura 5.4: Tempo de execução da aplicações de redução LU, com a arquitetura FairCPU em execução. Neste teste, as MVs estavam configuradas para utilizar 1 UP por VCPU.



	1 VCPU	2 VCPUs	4 VCPUs
C2D	1592,7	838,7	
Ci7	1574,4	798,7	399,9
Ci5	1650,4	843,8	420,9
X4	1676,1	836,2	420,8

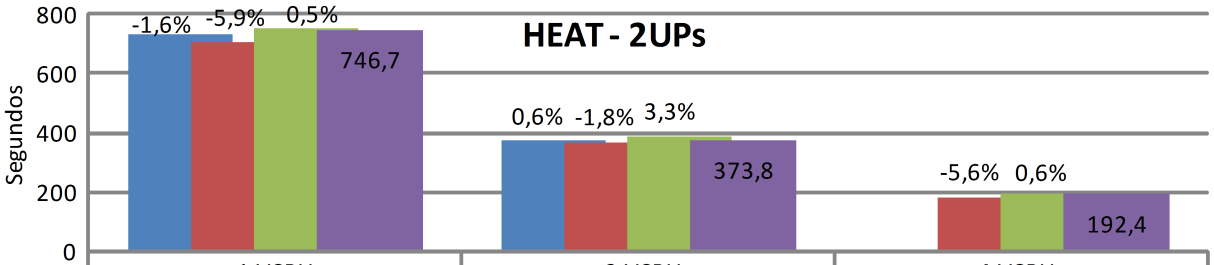
Figura 5.5: Tempo de execução da aplicação de transferência de calor, com a arquitetura FairCPU em execução. Neste teste, as MVs estavam configuradas para utilizar 1 UP por VCPU.

Para finalizar, as mesmas aplicações foram executadas em MVs com 1, 2 e 4 VCPUs, só que configuradas para utilizar 2 UPs por VCPU, ficando com 2, 4 e 8 UPs, respectivamente. Os resultados podem ser vistos nas Figuras 5.6 e 5.7.



	1 VCPU	2 VCPUs	4 VCPUs
C2D	955,1	474,2	
Ci7	968,3	499,3	245,7
Ci5	964,8	484,4	247,2
X4	960,3	482,3	244,7

Figura 5.6: Tempo de execução da aplicação de redução LU, com a arquitetura FairCPU em execução. Neste teste, as MVs estavam configuradas para utilizar 2 UPs por VCPU.



	1 VCPU	2 VCPUs	4 VCPUs
C2D	734,1	376,1	
Ci7	702,6	366,9	181,5
Ci5	750,9	386,2	193,6
X4	746,7	373,8	192,4

Figura 5.7: Tempo de execução da aplicação de transferência de calor, com a arquitetura FairCPU em execução. Neste teste, as MVs estavam configuradas para utilizar 2 UPs por VCPU.

Na aplicação de redução LU, a diferença mais significativa entre os tempos de execução foi 3.5% (no caso em que 2 VCPUs eram utilizadas), já para a aplicação de transferência de calor foi 5.9% (1 VCPU). Esta etapa confirma os resultados da etapa anterior, mostrando a eficiência da arquitetura FairCPU em prover poder computacional de forma mais homogênea, mesmo com as MVs sendo executadas em MFs bem diferentes.

5.2.3 Experimento 3 - FairCPU com carga de trabalho extra (sobrecarga da CPU)

O objetivo deste experimento é avaliar o impacto da carga de trabalho da MF no desempenho das aplicações em execução nas MVs. Diferente do testes realizados no experimento

anterior, onde as MVs eram executadas sozinhas nas MFs, este experimento foi executado para avaliar o isolamento de desempenho da arquitetura FairCPU, nos casos de extremo consumo de CPU por parte das MVs.

Para isso, duas MVs foram executadas ao mesmo tempo em cada MF. Uma delas executando a aplicação de transferência de calor e a outra executando o algoritmo de redução LU. Dois casos foram testados: MVs com 2 VCPUs e 2 UPs por VCPU, resultado em 8 UPs (sendo 4 UPs para cada MV); e MVs com 4 VCPUs e 1 UP por VCPU, também resultando em 8 UPs (4 UPs para cada MV). Neste experimento, a MF do tipo C2D não foi utilizada por possuir apenas 5 UPs.

O resultado deste experimento para a configuração 2VCPUS-2UPs pode ser visto na Figura 5.8, enquanto o resultado para a configuração 4VCPUs-1UP é apresentado na Figura 5.9. Os casos marcados com "- carga extra" representam os testes em que duas MVs eram executadas ao mesmo tempo na mesma MF, representando a carga de trabalho extra (concorrência pelo processador). Os resultados foram comparados com alguns dos resultados do Experimento 2, em que apenas uma MV era usada.

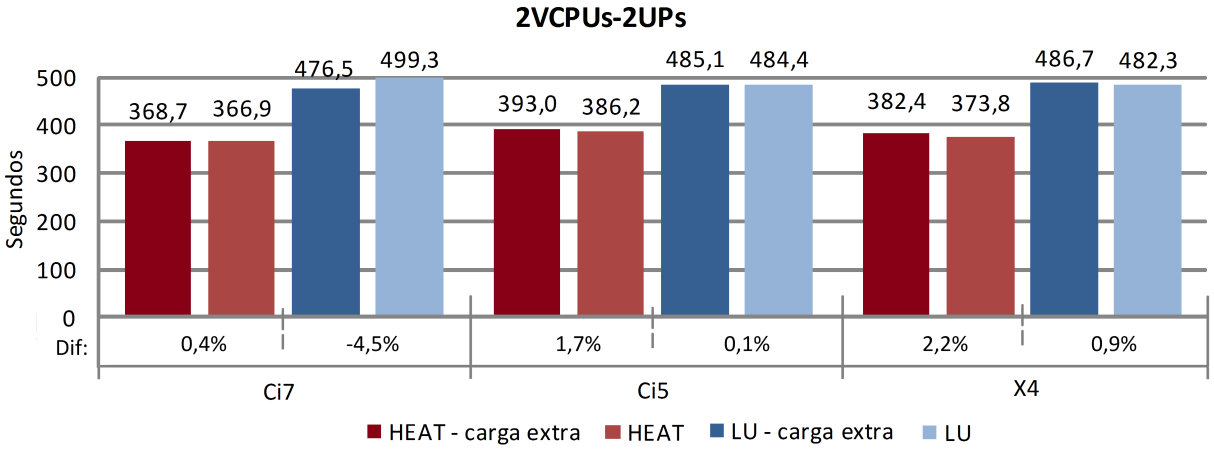


Figura 5.8: Comparação entre o tempo de execução das aplicações com e sem carga de trabalho extra para a configuração 2VCPUS-2UPs.

Pode ser observado que a maior diferença entre os resultados foi de 4,5%, na MF do tipo Ci7, para ambas as configurações (2VCPUS-2UPs e 4VCPUs-1UP). Era esperado um tempo de execução igual ou pouco maior para os testes com carga de trabalho extra, entretanto, o tempo de execução, principalmente para o problema de redução LU com 2VCPUs-2UPs, foi inesperadamente mais rápido.

Este comportamento parece ter sido causado pela tecnologia Turbo Boost, da Intel, que aumenta a velocidade de processadores com base na demanda de operação. Como, neste caso, cada uma das MVs usava 2 VCPUs, a perda de desempenho causada pela concorrência foi baixa, se comparada ao ganho de desempenho causado pela tecnologia Turbo Boost. Este comportamento mostra o quanto é difícil e desafiador trabalhar com ambientes heterogêneos, e apesar disso, os resultados mostram que a solução proposta está trabalhando bem e provendo um poder de computação mais homogêneo, o que pode ser provado pelos tempos de execução bem similares.

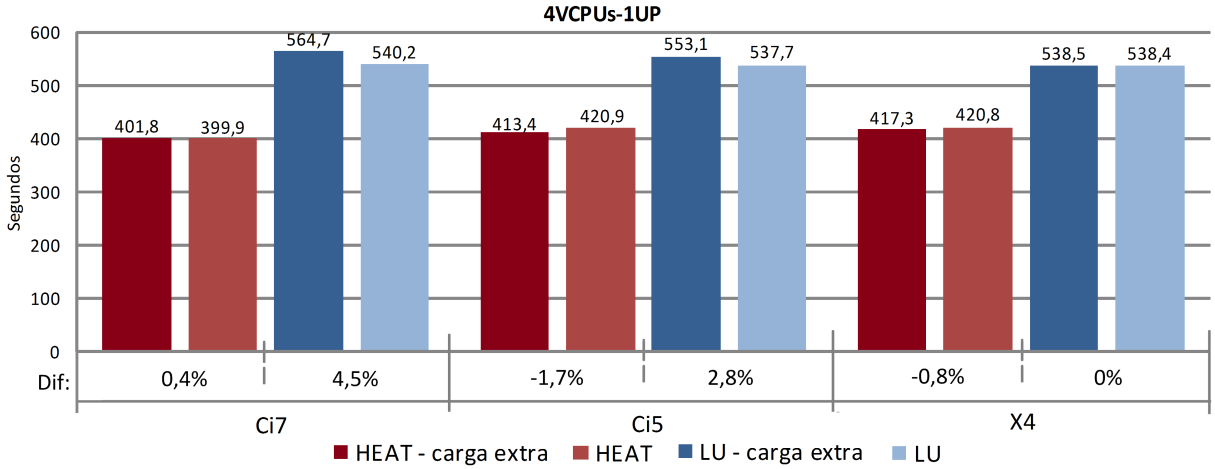


Figura 5.9: Comparação entre o tempo de execução das aplicações com e sem carga de trabalho extra para a configuração 4VCPUs-1UP.

5.2.4 Experimento 4 - Comparação da variação de desempenho com a Amazon EC2

Diversos estudos têm alertado para a variação de desempenho das MVs em execução na Amazon EC2, apesar deles utilizarem a ECU para representar o poder de processamento para cada uma de suas instâncias. A Tabela 5.3 apresenta as instâncias mais comuns da Amazon e a quantidade de ECUs de cada uma, além de outros detalhes.

Tabela 5.3: Tipos de instâncias mais comuns da Amazon EC2.

Tipo	VCPU (ECUs)	Memória	Arquitetura
m1.small	1 (1)	1.7 GB	32
m1.large	2 (4)	7.5 GB	64
m1.xlarge	2 (8)	15.0 GB	64
c1.medium	2 (5)	1.7 GB	32
c1.xlarge	8 (20)	7.0 GB	64

Neste experimento, a aplicação de transferência de calor e a de redução LU foram executadas em dois tipos de instâncias da Amazon, *m1.small* e *c1.medium*, em diferentes horários e zonas, durante 3 dias. Estas duas instâncias foram escolhidas porque a versão do Ubuntu Server 11.04, usada nos outros experimentos, é 32 bits, e SOs 32 bits só podem ser executados nestes tipos de instâncias.

Para cada teste, as aplicações foram executadas 10 vezes e, após a execução, as MVs foram desligadas, o que forçava a Amazon a criar e alocar novas instâncias. Usando esta metodologia, as MVs poderiam ser alocadas em diferentes MFs, com diferentes tipos de processadores. A Tabela 5.4 e a Tabela 5.5 apresentam os detalhes sobre cada execução para instâncias do tipo *m1.small* e *c1.medium*, respectivamente. São apresentados os horários em que as MVs foram inicializadas, as zonas em que elas foram executadas e, principalmente, o processador da MF em que as MVs foram alocadas[4]. As tabelas confirmam a heterogeneidade das MFs que compõem o *datacenter* da Amazon.

[4]O processador não pode ser escolhido pelo cliente, pois depende da MF, e é a Amazon que cuida da alocação.

Tabela 5.4: Detalhes sobre cada execução das aplicações para instâncias do tipo *m1.small*.

Instância	Zona	Processador da MF	Horário (GMT -3)
us-1	*US-east* (Virgínia)	Intel Xeon E5430 @ 2,66 GHz	1:30 am
us-2	*US-east* (Virgínia)	AMD Opteron 2218 HE @ 2,60 GHz	11:30 am
us-3	*US-east* (Virgínia)	Intel Xeon E5645 @ 2,40 GHz	8:00 am
us-4	*US-east* (Virgínia)	AMD Opteron 2218 HE @ 2,60 GHz	8:50 pm
us-5	*US-east* (Virgínia)	Intel Xeon E5507 @ 2,27 GHz	9:00 pm
eu-1	*EU* (Irlanda)	Intel Xeon E5507 @ 2,27 GHz	1:30 am
eu-2	*EU* (Irlanda)	Intel Xeon E5430 @ 2,66 GHz	12:30 pm
eu-3	*EU* (Irlanda)	Intel Xeon E5430 @ 2,66 GHz	8:50 am
eu-4	*EU* (Irlanda)	Intel Xeon E5507 @ 2,27 GHz	9:00 pm
eu-5	*EU* (Irlanda)	Intel Xeon E5645 @ 2,40 GHz	11:30 am
ap-1	*Asia-pacific* (Singapura)	Intel Xeon E5645 @ 2,40 GHz	8:00 pm
ap-2	*Asia-pacific* (Singapura)	Intel Xeon E5645 @ 2,40 GHz	8:50 am
ap-3	*Asia-pacific* (Singapura)	Intel Xeon E5645 @ 2,40 GHz	9:00 pm
sa-4	*Asia-pacific* (Singapura)	Intel Xeon E5645 @ 2,40 GHz	1:30 am
sa-1	*South America* (Brasil)	Intel Xeon E5645 @ 2,40 GHz	11:30 am
sa-2	*South America* (Brasil)	Intel Xeon E5645 @ 2,40 GHz	8:00 pm
sa-3	*South America* (Brasil)	Intel Xeon E5645 @ 2,40 GHz	9:00 pm
sa-4	*South America* (Brasil)	Intel Xeon E5645 @ 2,40 GHz	1:30 pm

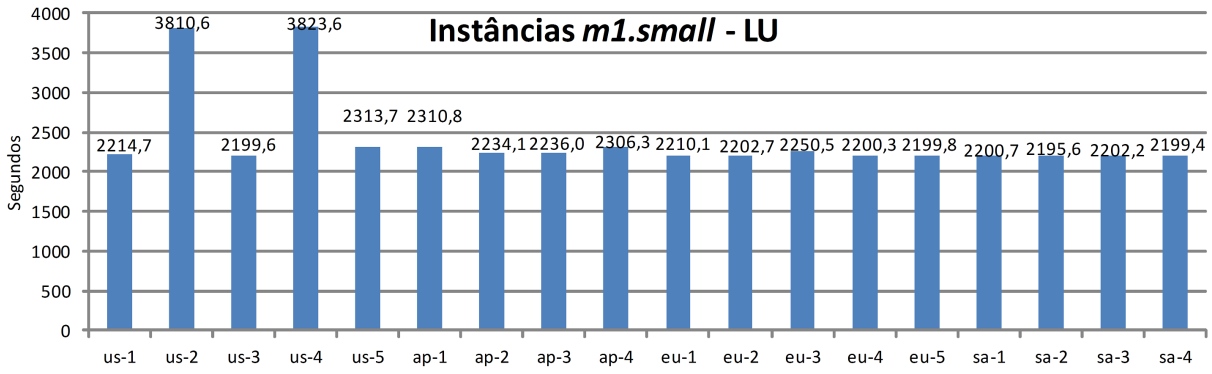


Figura 5.10: Tempo de execução da aplicação de redução LU para instâncias da Amazon EC2 do tipo *m1.small*.

A Figura 5.10 mostra os resultados para execução da aplicação de redução LU em instâncias do tipo *m1.small*, enquanto a Figura 5.11 mostra os resultados para a aplicação de transferência de calor. O tempo de execução para os testes us-2 e us-4 foi aproximadamente 65% maior do que os outros, para o algoritmo de redução LU, e 53% maior para o problema de transferência de calor. Esta grande variação deve ter sido causada pelo processador da MF onde a MV foi alocada, AMD Opteron Processor 2218 HE, nos dois casos. Desconsiderando os testes com este processador, a maior diferença entre os tempos de execução foi 5,1% para o problema de redução LU (entre us-3 e us-5) e 8,1% para a aplicação de transferência de calor (entre us-1 e us-4).

Os resultados para instâncias do tipo *c1.medium* são apresentados nas Figuras 5.12 e 5.13. A variação do tempo de execução não foi tão grande quanto a das instâncias do tipo *m1.small*, uma vez que a maior diferença entre os tempos de execução foi de 20,5% para redução LU (entre us-8 e eu-8) e 12,8% para transferência de calor (entre us-8 e eu-10).

Tabela 5.5: Detalhes sobre cada execução das aplicações para instâncias do tipo *c1.medium*.

Instância	Zona	Processador da MF	Horário (GMT -3)
us-6	*US-east* (Virgínia)	Intel Xeon E5506 @ 2,13 GHz	1:30 am
us-7	*US-east* (Virgínia)	Intel Xeon E5506 @ 2,13 GHz	11:30 am
us-8	*US-east* (Virgínia)	Intel Xeon E5410 @ 2,33 GHz	8:00 am
us-9	*US-east* (Virgínia)	Intel Xeon E5506 @ 2,13 GHz	8:50 am
us-10	*US-east* (Virgínia)	Intel Xeon E5506 @ 2,13 GHz	9:00 pm
eu-6	*EU* (Irlanda)	Intel Xeon E5410 @ 2,33 GHz	1:30 am
eu-7	*EU* (Irlanda)	Intel Xeon E5410 @ 2,33 GHz	12:30 pm
eu-8	*EU* (Irlanda)	Intel Xeon E5410 @ 2,33 GHz	8:50 am
eu-9	*EU* (Irlanda)	Intel Xeon E5410 @ 2,33 GHz	9:00 pm
eu-10	*EU* (Irlanda)	Intel Xeon E5506 @ 2,13 GHz	11:30 am
ap-5	*Asia-pacific* (Singapura)	Intel Xeon E5410 @ 2,33 GHz	8:00 pm
ap-6	*Asia-pacific* (Singapura)	Intel Xeon E5506 @ 2,13 GHz	1:30 am
ap-7	*Asia-pacific* (Singapura)	Intel Xeon E5410 @ 2,33 GHz	8:50 am
ap-8	*Asia-pacific* (Singapura)	Intel Xeon E5410 @ 2,33 GHz	9:00 pm
sa-5	*South America* (Brasil)	Intel Xeon E5506 @ 2,13 GHz	8:00 pm
sa-6	*South America* (Brasil)	Intel Xeon E5506 @ 2,13 GHz	1:30 am
sa-7	*South America* (Brasil)	Intel Xeon E5506 @ 2,13 GHz	8:50 am
sa-8	*South America* (Brasil)	Intel Xeon E5506 @ 2,13 GHz	9:00 pm

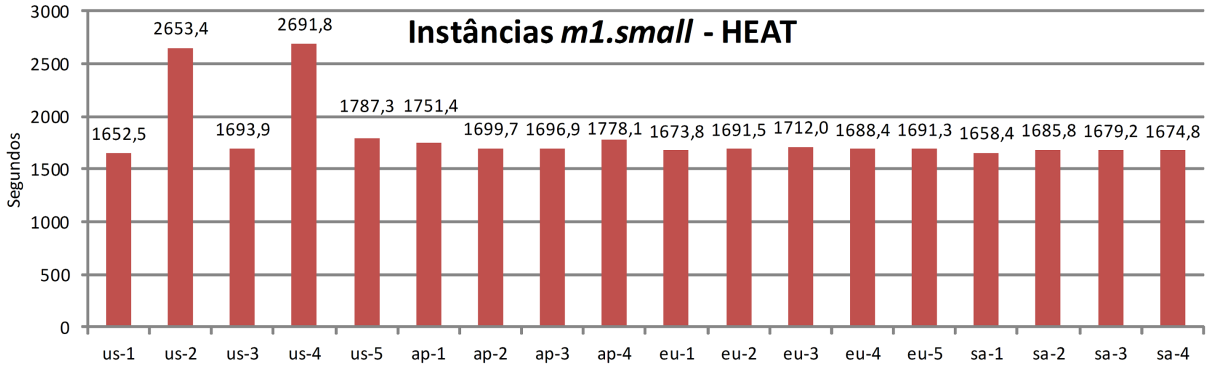


Figura 5.11: Tempo de execução da aplicação de transferência de calor para instâncias da Amazon EC2 do tipo *m1.small*.

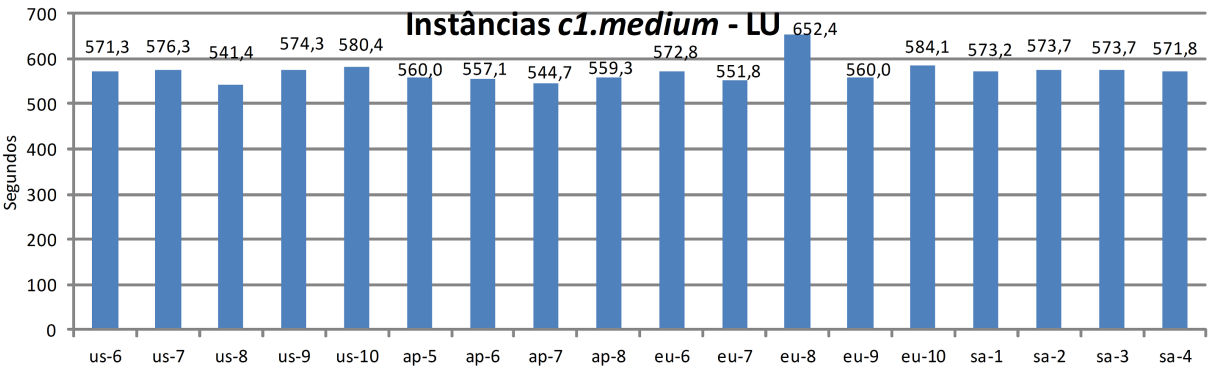


Figura 5.12: Tempo de execução da aplicação de redução LU para instâncias da Amazon EC2 do tipo *c1.medium*.

Este experimento confirma que a Amazon EC2 tem problemas de variação de desempenho, principalmente para instâncias do tipo *c1.small*, que são frequentemente solicitadas

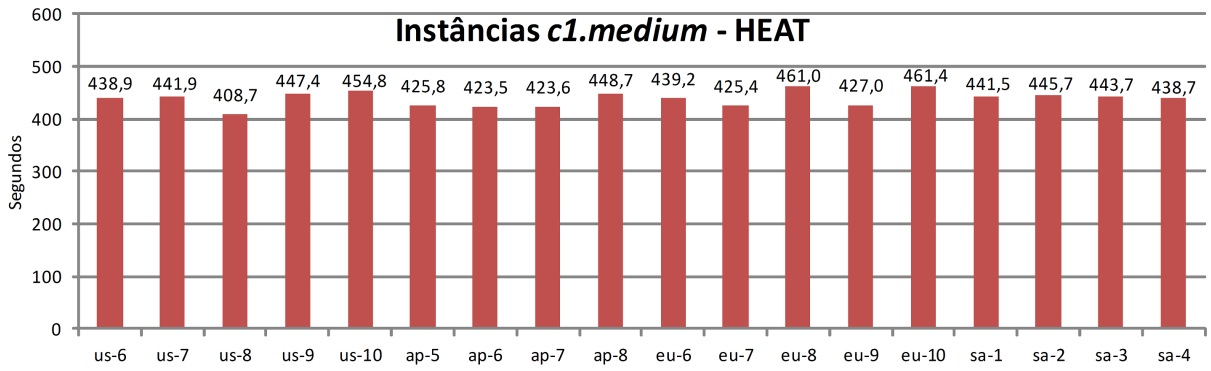


Figura 5.13: Tempo de execução da aplicação de transferência de calor para instâncias da Amazon EC2 do tipo *c1.medium*.

pelos clientes (SCHAD; DITTRICH; QUIANÉ-RUIZ, 2010). Os resultados também mostram que algumas zonas são mais estáveis do que outras, porém, o fato dos testes terem sido realizados em poucos dias nos impossibilita de ter uma visão completa.

Outra descoberta interessante é que o poder de processamento da ECU é parecido com o valor que foi definido para a UP (aproximadamente 50% do *core* do Xeon). Este fato pode ser observado ao comparar o tempo de execução para instâncias do tipo *m1.small* na Amazon (Figuras 5.10 e 5.11) e os testes com 1VCPU-1UP (Figuras 5.4 e 5.5) na nossa nuvem privada.

Infelizmente, converter o poder de processamento de ECUs para UPs não é uma tarefa fácil. As métricas utilizadas pela Amazon para configurar a ECU não são conhecidas, e, para dificultar um pouco o trabalho, a Amazon utilizada uma versão customizada do hipervisor Xen, que deve ter passado por diversas otimizações. Estas diferenças podem levar a resultados inesperados. Um destes resultados pode ser visto ao comparar o tempo de execução das instâncias *c1.medium* da Amazon, que tem 2 VCPUs e 5 ECUs, com alguns dos testes da Seção 5.2.2, com 2VCPUs-2UPs (Figura 5.8). O tempo de execução para 2VCPUs-4UPs é menor do que na Amazon, sendo que era esperado o contrário, uma vez que 5 ECUS deveriam ter um poder computacional maior do que 4 UPs, já que foram notadas semelhanças entre os casos 1VCPU-1ECU e 1VCPU-1UP. Essa constatação abre espaço para trabalhos futuros, principalmente em nuvens híbridas, ao explorar a relação "ECU-UP"e a execução de aplicações na nuvem privada local e na nuvem pública comercial.

5.2.5 Experimento 5 - Execução de aplicações MPI

O objetivo deste experimento é avaliar o impacto da utilização das UPs na execução da aplicação de multiplicação de matrizes (paralelizada com MPI), que, além de utilizar processamento, utiliza recursos de rede. Outra diferença significativa com relação aos outros experimentos é que a execução da aplicação envolve mais de uma MF, pois é executada em um *cluster* virtual.

Para este experimento, a arquitetura FairCPU foi integrada com o *framework* Nebulous (GALANTE; BONA, 2011), que automatiza a execução de aplicações OpenMP e MPI em

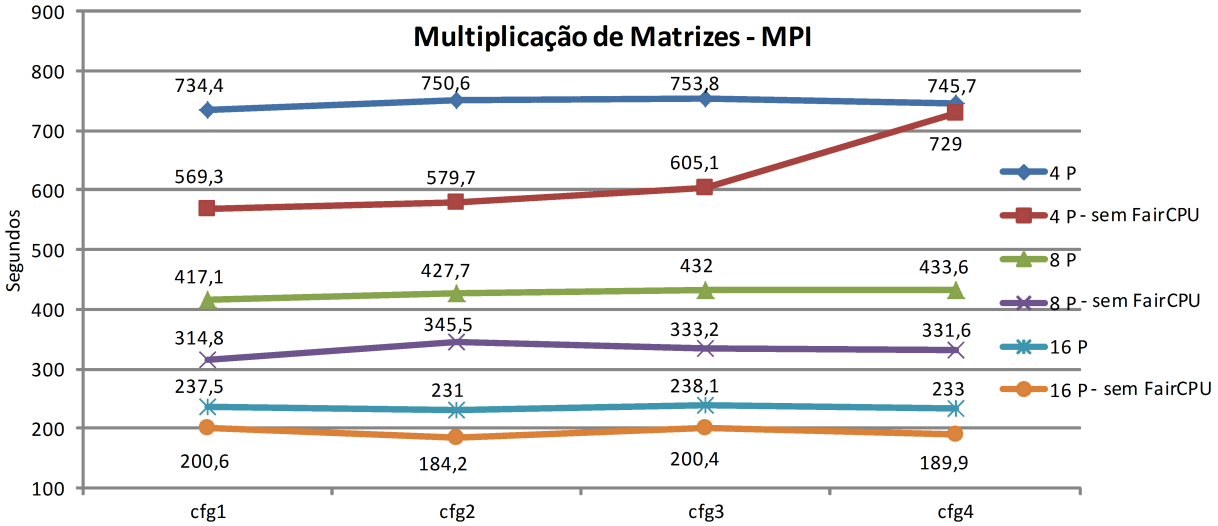


Figura 5.14: Tempo de execução para a aplicação de multiplicação de matrizes (paralelizada com MPI) com o módulo Gerenciador de Limites habilitado e desabilitado.

ambientes virtuais, se responsabilizando por criar automaticamente os *clusters* virtuais.

Foram lançados testes com 4, 8 e 16 processos MPI, cada um deles sendo executado individualmente em uma MV com 1 VCPU e 2 UPs. Para cada caso, quatro configurações com diferentes MFs foram utilizadas. As configurações são apresentadas na Tabela 5.6, onde o número entre colchetes ([ ]) representa a quantidade de MVs que foram alocadas na MF (exemplo, Ci5[2] significa que foram alocadas 2 MVs na MF do tipo Ci5). Em todos os testes com 16 processos, foram utilizadas quatro MVs por MF.

Tabela 5.6: Relação da quantidade de processos por MF para cada configuração do experimento com a aplicação MPI.

	Processos		
Configurações	4	8	16
cfg1	Ci7[1] Ci5[1] Ci5[1] X4[1]	Ci7[3] Ci5[3] Ci5[2]	Ci7[4] Ci5[4] Ci5[4] X4[4]
cfg2	Ci5[2] Ci5[2]	Ci5[4] Ci5[4]	Ci7[4] Ci5[4] Ci5[4] X4[4]
cfg3	Ci7[2] Ci5[1] X4[1]	Ci7[3] Ci5[3] X4[2]	Ci7[4] Ci5[4] Ci5[4] X4[4]
cfg4	X4[4]	Ci5[3] Ci5[3] X4[2]	Ci7[4] Ci5[4] Ci5[4] X4[4]

A Figura 5.14 apresenta o tempo de execução das aplicações com e sem a FairCPU em execução. Os resultados mostram uma diferença de 28% entre a execução mais rápida e a mais lenta, ao executar a multiplicação de matrizes com 4 processos (4 P) e a FairCPU desabilitada. Entretanto, essa grande variação foi causada pelo alto tempo de execução para a configuração cfg4, em que todas as MVs estavam sendo executadas na mesma MF.

Como as aplicações MPI utilizam muita comunicação através da rede, o fato das quatro MVs estarem sendo executadas na mesma MF pode ter causado uma sobrecarga na rede, o que acarreta um aumento no tempo de execução da aplicação. O hipervisor KVM tem alguns problemas conhecidos de desempenho para comunicação de rede, que se tornam mais relevantes quando a rede é sobrecarregada. Como trabalho futuro, será investigada a utilização de *drivers virtio* (RUSSELL, 2008) para melhorar o desempenho de I/O de rede e disco das MVs e, por

conseguinte, melhorar o desempenho de aplicações MPI.

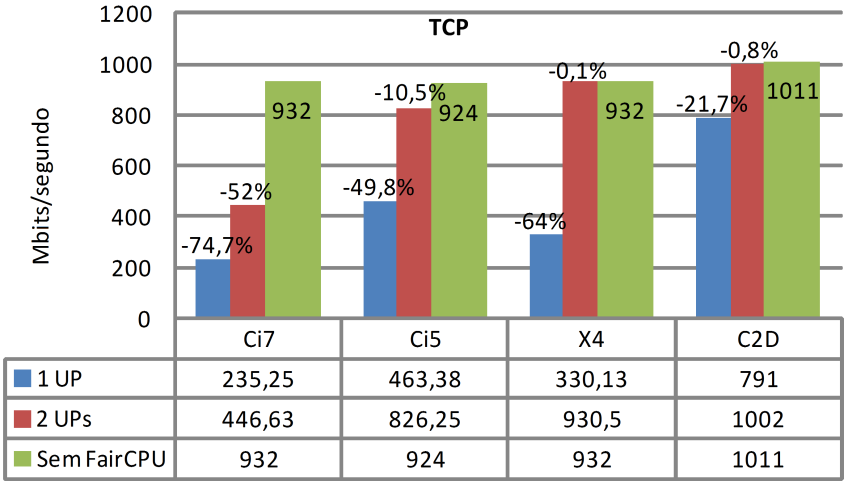
Desconsiderando o tempo de execução da configuração cfg4, a maior variação foi de 8,5% com o Gerenciador de Limites desabilitado (16 processos). Já para os testes com todos os módulos da FairCPU habilitados, a maior variação foi de 4% (8 processos).

O desempenho da rede parece ser um fator bastante decisivo para definir o tempo de execução dessa aplicação. Para que a arquitetura FairCPU traga vantagens maiores, é preciso um estudo mais aprofundado sobre o comportamento das aplicações MPI. De qualquer forma, a utilização da arquitetura deixou o resultado mais homogêneo, apesar de o tempo de execução ter aumentado consideravelmente. Não se pode esquecer que, no caso com a FairCPU habilitada, algumas das MFs (Ci5 e Ci7) ainda teriam UPs livres para alocar outras MVs, ao contrário do caso com a FairCPU desabilitada, onde as MFs já estariam utilizando todo seu poder de processamento.

5.2.6 Experimento 6 - Impacto da UP no I/O de Rede

O objetivo deste experimento é avaliar o impacto da utilização das UPs no I/O de rede. Como a arquitetura FairCPU limita o uso da CPU das MVs para prover o poder de processamento baseada na quantidade de UPs requisitadas, é importante descobrir qual o impacto dessa solução no I/O de rede.

O *benchmark* Iperf foi executado em uma MV com 1 e 2 UPs, e o resultado foi comparado com a execução na mesma MV, sem a arquitetura FairCPU estar em execução. A Figura 5.15 e Figura 5.16 apresentam os resultados para vazão de rede com fluxos de dados TCP e UDP, respectivamente.

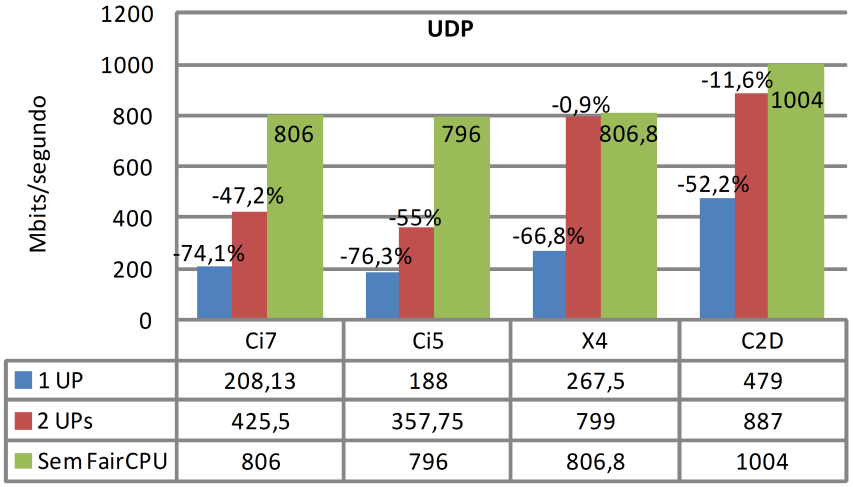


	Ci7	Ci5	X4	C2D
1 UP	235,25	463,38	330,13	791
2 UPs	446,63	826,25	930,5	1002
Sem FairCPU	932	924	932	1011

Figura 5.15: Vazão de rede para fluxo de dados TCP, com a MV sujeita a diferentes limites de uso da CPU.

É possível observar que, em todos os casos, há uma perda de desempenho, principalmente para as MVs com 1 UP, onde a perda de desempenho chegou a 76,3%. Este resultado era esperado e está de acordo com trabalhos presentes na literatura. É possível notar também o melhor desempenho das MVs que foram executadas com o hipervisor Xen, na MF do tipo C2D.

Este resultado também era esperado, uma vez que é sabido que as MVs paravirtualizadas têm um melhor desempenho no I/O de rede, se comparada à virtualização assistida por *hardware*.



	Ci7	Ci5	X4	C2D
1 UP	208,13	188	267,5	479
2 UPs	425,5	357,75	799	887
Sem FairCPU	806	796	806,8	1004

Figura 5.16: Vazão de rede para fluxo de dados UDP, com a MV sujeita a diferentes limites de uso da CPU.

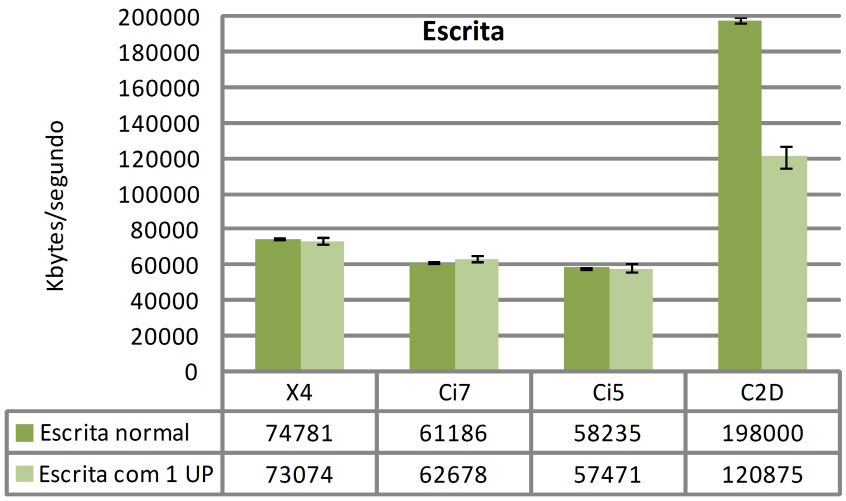
Por fim, é possível notar que a vazão alcançada para conexões TCP é maior do que a alcançada para UDP. Esse comportamento é percebido principalmente nas MVs que foram executadas nas MFs Ci7, Ci5 e X4, que usam KVM.

Este experimento apresenta resultados importantes e mostra uma desvantagem da arquitetura FairCPU. A execução de aplicações que necessitam de recursos de rede pode ficar comprometida, principalmente as aplicações de uso intensivo de rede (*network-intensive*) e as de computação intensiva de dados (*data-intensive*), que geralmente acessam os dados em um *storage* através da rede. Porém, essas perdas de desempenho estão mais relacionadas ao hipervisor e ao tipo de virtualização utilizada, o que pode ser contornado futuramente com melhorias nas próximas versões desses *software*. Outro detalhe importante é que a maior perda de desempenho foi notada em MVs com apenas 1 UP, e espera-se que as aplicações que utilizam muito a rede também precisem de muito processamento, tendo que solicitar mais de 1 UP, o que já melhoraria muito o desempenho.

5.2.7 Experimento 7 - Impacto da UP no I/O de Disco

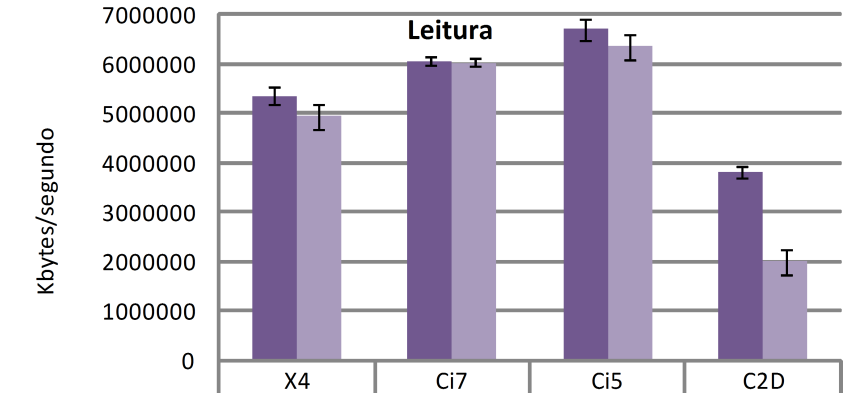
Este experimento tem o objetivo de avaliar o impacto da utilização das UPs no I/O de disco. Como a arquitetura FairCPU limita o uso da CPU das MVs para prover o poder de processamento baseada na quantidade de UPs requisitadas, é importante descobrir qual o impacto dessa solução na escrita e leitura de disco.

O *benchmark* IOzone foi executado em uma MV com sistema de arquivos ext3 (*third extended file system*) e 1 UP. O *benchmark* foi configurado para escrever e ler um arquivo de 500 Mbytes e o resultado foi comparado com a execução na mesma MV, com a FairCPU desabilitada. A Figura 5.17 e Figura 5.18 apresentam os resultados para escrita e leitura no disco, respectivamente.



	X4	Ci7	Ci5	C2D
Escrita normal	74781	61186	58235	198000
Escrita com 1 UP	73074	62678	57471	120875

Figura 5.17: Desempenho de escrita no sistema de arquivos ext3, com a MV sujeita a diferentes limites de uso da CPU.



	X4	Ci7	Ci5	C2D
Leitura normal	5362561	6054434	6699613	3808656
Leitura com 1 UP	4946815	6012244	6348315	2001622

Figura 5.18: Desempenho de leitura do sistema de arquivos ext3, com a MV sujeita a diferentes limites de uso da CPU.

Observa-se que a limitação do uso da CPU praticamente não interferiu no resultado para as MVs que foram executadas com o hipervisor KVM, já a MV que foi executada na MF C2D, com Xen, sofreu uma perda de desempenho considerável para o caso com 1 UP. Além disso, percebe-se que o hipervisor Xen tem um melhor desempenho para escrita, enquanto o hipervisor KVM tem um melhor desempenho para leitura.

Este resultado é importante para aplicações de computação intensiva de dados e pode ser melhor explorado em trabalhos futuros, ao utilizar outros *benchmarks* e outros sistemas de arquivo, como ext4, ReiserFS e NFS.

5.2.8 Experimento 8 - Alteração dinâmica da quantidade de UPs

O objetivo deste experimento é apresentar como as funcionalidades de migração de MVs e alteração dinâmica da quantidade de UPs podem ser utilizadas, bem como o compor-

tamento das MFs durante esses eventos. Para isso, uma VM com 2 VCPUs, 3 UPs e 2 GB de memória foi criada e alocada numa MF do tipo X4. O experimento consiste na execução do *benchmark* Linpack na MV, enquanto um *script*[5] (Algoritmo 3) envia comandos, tanto à FairCPU, para alterar a quantidade de UPs, quanto ao *middleware* OpenNebula, para solicitar a migração da MV para uma MF do tipo Ci5.

Algoritmo 3: *Script* utilizado para alterar quantidade de UPs e migrar a MV.

```
1 #!/bin/bash
2 sleep 80
3 ./fclient.rb setpu 209 2
4 sleep 80
5 onevm livemigrate 209 4
6 sleep 80
7 ./fclient.rb setpu 209 3
```

Durante toda a execução do experimento, o uso de CPU (valor total e valor para cada núcleo) das MFs estava sendo monitorado e coletado. As Figuras 5.19 e 5.20 apresentam os resultados do monitoramento e mostram como a carga de CPU das MFs X4 e Ci5 variou com o tempo, respectivamente. Diferente dos outros experimentos, onde o Linpack era executado 30 vezes sequenciais e só a média dos resultados era apresentada, neste, estamos interessados na variação do resultado do *benchmark* com o tempo, uma vez que estaremos alterando a quantidade de UPs da MV (ou seja, o poder de processamento). Os resultados do Linpack e sua variação com o tempo são apresentados na Figura 5.21.

O Algoritmo 3 apresenta o *script* que foi utilizado. Durante 80 segundos, o Linpack foi executado com 3 UPs na MF X4, e três resultados, praticamente iguais a 9 GFLOPS (3 UPs), foram retornados. Pode-se observar que o uso de CPU total desta MF (Figura 5.19) é, aproximadamente, 150%, exatamente o valor apresentado na Tabela 5.2.

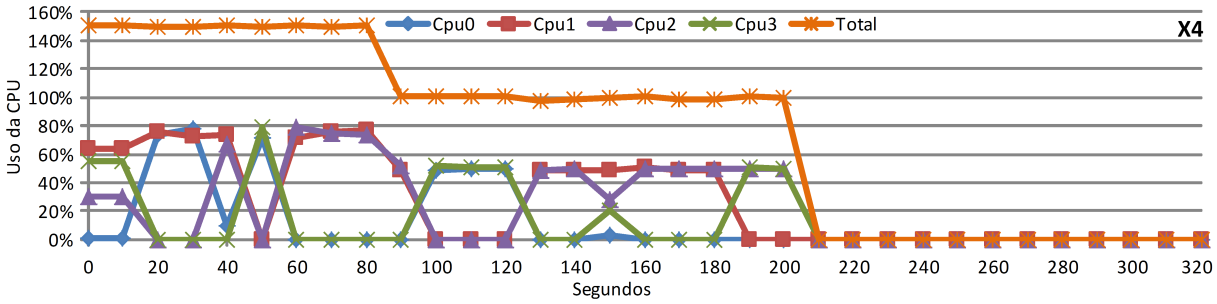


Figura 5.19: Monitoramento da CPU da MF X4.

Na linha 3, é solicitada à FairCPU a alteração da quantidade de UPs da MV (cujo identificador é 209) para 2. Pode-se observar que a utilização da CPU da MF X4 cai para 100% e o Linpack retorna dois resultados. O segundo é igual 6,1 GLOPS, valor esperado para 2 UPs, enquanto o primeiro é igual a 7,2 GFLOPS, cuja explicação para esse valor elevado é o fato do Linpack ter iniciado sua execução ainda com 3 UPs.

[5] O *script* foi executado no controlador, que é a MF onde o OpenNebula e a FairCPU estavam em execução.

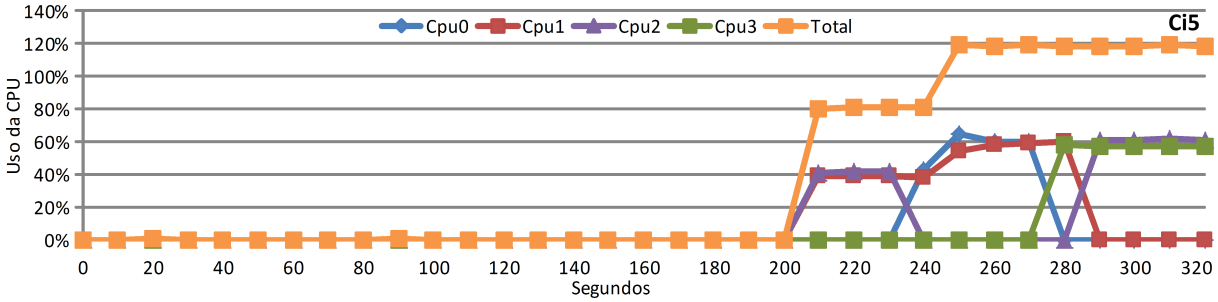


Figura 5.20: Monitoramento da CPU da MF Ci5.

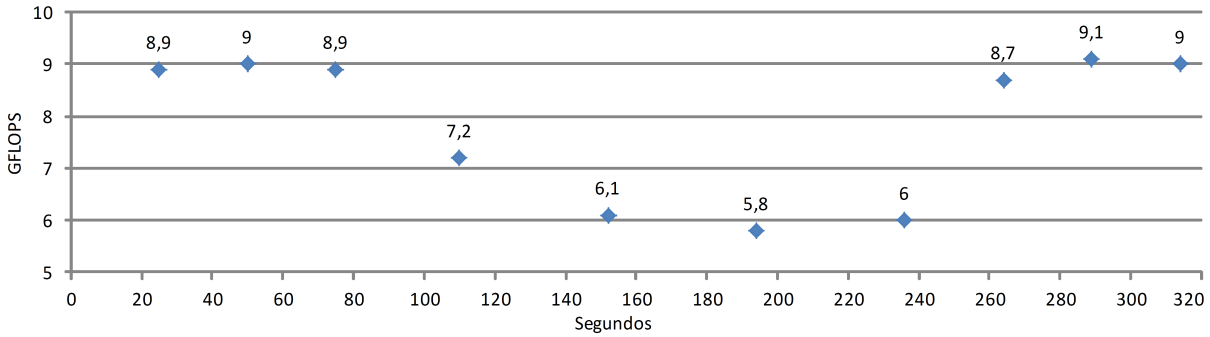


Figura 5.21: Resultado do *benchmark* Linpack durante a execução do *script*.

Passados mais 80 segundos, a linha 5 solicita ao OpenNebula que a MV seja migrada para uma MF do tipo Ci5, cujo identificador é 4. Foi utilizada a migração quente (*livemigration*), onde a MV é transferida entre as MFs sem interrupção na execução. As Figuras 5.19 e 5.20 mostram que a migração durou entre 40 e 50 segundos, uma vez que o processador da MF Ci5 começou a ser bastante utilizado entre os segundos 200 e 210. Durante esse processo, o Linpack retornou dois valores, o segundo foi 6 GFLOPS (esperado para 2 UPs) e 5,8 GFLOPS, um pouco abaixo do valor esperado para 2 UPs, mas pode ser explicado pela sobrecarga do processo de migração, que ocorre exatamente entre os segundos 160 e 210.

Por fim, a linha 7 solicita à FairCPU a alteração da quantidade de UPs da MV para 3. A Figura 5.20 mostra que o uso de CPU da máquina Ci5 passou de, aproximadamente, 80% para, aproximadamente, 120%, o que está em conformidade com a Tabela 5.2. Durante esse período, o Linpack retornou três valores, dos quais apenas o primeiro está um pouco abaixo do esperado, pelo fato da execução ter começado ainda com 2 UPs.

Este experimento mostra como a alteração dinâmica de UPs pode ser útil, bem como a eficiência da arquitetura FairCPU em lidar com esta operação e a migração de MVs.

5.2.9 Experimento 9 - Escalonamento

Este experimento visa avaliar a utilização da política de escalonamento, apresentada na Seção 4.4.3, para redução do consumo de energia do *datacenter*. A Tabela 5.7 apresenta o consumo de energia por UP, para cada MF da nossa nuvem privada, medido com o wattímetro.

Como a arquitetura FairCPU utiliza UPs e divide o processador em várias unidades,

Tabela 5.7: Tabela de consumo de energia por UP para cada MF da nuvem privada. O consumo é medido em Watts.

	idle	**1 UP**	**2 UPs**	**3 UPs**	**4 UPs**	**5 UPs**	**6 UPs**	**7 UPs**	**8 UPs**	**9 UPs**	**10 UPs**
Ci7	136 W	165 W	186 W	200 W	211 W	223 W	235 W	247 W	260 W	271 W	281 W
Ci5	45 W	60 W	71 W	83 W	94 W	105 W	116 W	130 W	146 W	157 W	NA
X4	79 W	83 W	93 W	102 W	113 W	124 W	134 W	143 W	152 W	NA	NA
C2D	67 W	77 W	89 W	100 W	110 W	118 W	NA	NA	NA	NA	NA

não seria justo comparar com técnicas de escalonamento que não utilizam. Por exemplo, uma MF do tipo Ci7 pode alocar até 10 MVs, se cada uma delas tiver apenas 1 UP, enquanto alocaria apenas 4, se fosse utilizado o número bruto de CPUs. Por isso, apenas as políticas suportadas pela FairCPU são testadas nesse experimento: *Packing*, *Striping*, *Load-aware*, Aleatória e de redução de energia (*Energy-aware*).

Diferente de todos os outros, este experimento foi realizado por simulação, utilizando as implementações em Ruby das heurísticas. Foi simulada a existência de 10 MFs do tipo Ci5, 6 do tipo Ci7, 6 do tipo X4 e 6 do tipo C2D. Dois testes diferentes foram realizados. No primeiro, 128 MVs (com 1 VCPU, 1UP e 1GB de memória) foram requisitadas para simular a criação de um *cluster* virtual com 128 nodos. No segundo, foi requisitada a criação de 36 MVs, sendo 6 para cada uma das configurações: 1 VCPU e 1 GB de memória, 1 VCPU e 2 GB de memória, 2 VCPUs e 2 GB de memória, 2 VCPUs e 4 GB de memória, 4 VCPUs e 4 GB de memória, 4 VCPUs e 8 GB de memória; todas com 1 UP por VCPU. Foi considerado que as MVs poderiam ser executadas em qualquer MF, desconsiderando a existência de hipervisores diferentes.

A Figura 5.22 apresenta o resultado[6] para o primeiro teste. Pode-se perceber que a heurística *Energy-aware* consome menos energia que as demais, e a economia de energia chega a 4,8%. A Figura 5.23 apresenta o resultado para o segundo teste e confirma que utilizando a política de escalonamento *Energy-aware* é possível reduzir o gasto de energia em um ambiente heterogêneo, mantendo o desempenho das MVs independente da MF subjacente. Para este teste, a redução do consumo de energia chegou a 7,1%.

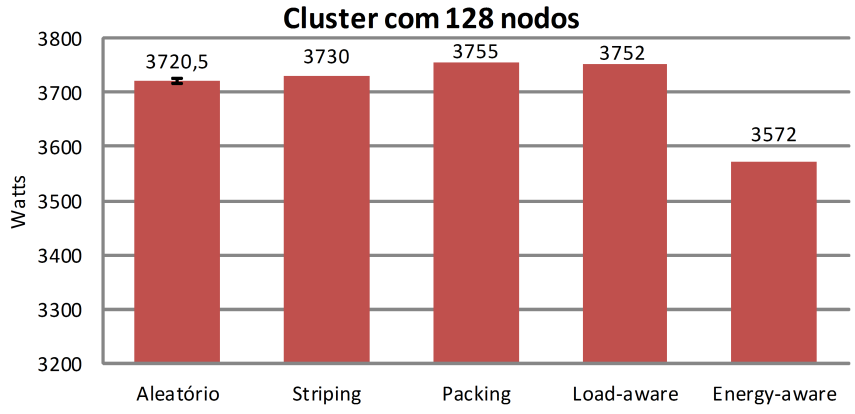


Figura 5.22: Consumo de energia para o teste com a criação de 128 MVs iguais.

[6]O valor para a heurística Aleatória é a média de 30 execuções, enquanto todas as outras heurística retornam sempre o mesmo valor.

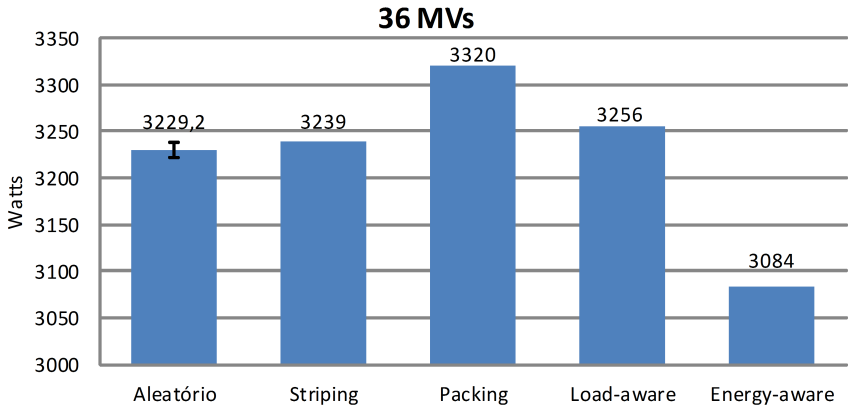


Figura 5.23: Consumo de energia para o teste com a criação de 36 MVs de 6 diferentes configurações.

As políticas implementadas servem para deixar o módulo Escalonador compatível com o *middleware* OpenNebula, uma vez que este disponibiliza algumas dessas políticas por padrão. Além disso, o presente trabalho abre diversas oportunidades de trabalhos futuros relacionados a escalonamento, pois a arquitetura fornece todas as funcionalidades necessárias para abstrair a heterogeneidade das MFs da infraestrutura, ao trabalhar com as UPs.

6 TRABALHOS RELACIONADOS

O presente trabalho apresentou diversas ideias, dentre as quais a principal é a utilização das unidades de processamento. Nenhum outro trabalho, no contexto de nuvens privadas, utilizou solução parecida para prover poder computacional de forma homogênea.

No contexto de nuvens públicas, a grande maioria dos provedores de IaaS utiliza a quantidade de GHz ou a quantidade de CPUs para representar o poder computacional das MVs. A Amazon é a única empresa que utiliza uma estratégia parecida com a das UPs, ao utilizar a ECU para informar aos clientes o poder de computação de suas instâncias. Apesar disso, ela enfrenta problemas de imprevisibilidade no desempenho.

Para facilitar a apresentação dos trabalhos relacionados, este capítulo está dividido em três seções. Primeiro, os trabalhos sobre análise de desempenho e que abordam a variabilidade de desempenho em nuvens comerciais são apresentados; segundo, são apresentados os trabalhos que utilizaram alguma técnica para limitar o uso da CPU e alterar dinamicamente os recursos alocadas para as MVs; por fim, trabalhos que abordam políticas de escalonamento são apresentados.

6.1 Análise e variabilidade de desempenho em ambientes virtualizados

Há diversos trabalhos que exploram a análise de desempenho de MVs em ambientes de computação em nuvem, utilizando diversos *benchmarks* e hipervisores. Deshane et al. (2008) apresentam uma análise quantitativa do Xen e KVM, comparando o desempenho geral ao apresentar resultados de *benchmarks* para isolamento de desempenho e escalabilidade. Os resultados, quanto ao desempenho da CPU, são praticamente iguais para Xen e KVM, o que é mais um ponto para justificar nossa escolha pelos dois hipervisores.

Keahey et al. (2007) analisam a utilidade dos serviços de computação em nuvem atuais para computação científica, ao fazer testes na plataforma Amazon EC2, usando um *benchmark* Linpack para avaliação. O Linpack também foi utilizado por Napper e Bientinesi (2009) para investigar se utilizando a Amazon EC2 era possível criar um *cluster* virtual que tivesse poder de processamento para entrar na lista TOP 500 (TOP500.ORG, 2012). Os experimentos apresentados indicaram que a nuvem da Amazon ainda não estava madura o suficiente para executar aplicações de CAD. Hoje, a Amazon tentou resolver esse problema ao criar instâncias especificas para CAD, ao lançar instâncias de *clusters* convencionais e *cluster* de GPUs (unidades de processamento gráfico).

Outros autores (OSTERMANN et al., 2010; VECCHIOLA; PANDEY; BUYYA, 2009) também avaliaram o desempenho da infraestrutura da Amazon para execução de aplicações científicas e CAD. Ostermann et al. (2010) foram além, ao relacionar o desempenho dos diferentes tipos de instâncias com os preços, calculando qual seria a relação GFLOPS/dólar para cada instância.

O problema de variabilidade de desempenho nos provedores de IaaS também foi

explorado. Schad, Dittrich e Quiané-Ruiz (2010) realizaram diversos experimentos na Amazon EC2, eles detectaram que a variação da vazão de rede teve uma perda de desempenho considerável, chegando a 54% para *upload* de dados, enquanto o tempo de execução de aplicações MapReduce chegou a variar 24% em alguns casos. Iosup, Yigitbasi e Epema (2011) analisaram o desempenho de diversos serviços do Google e da Amazon, e detectaram uma grande variabilidade de desempenho, com vários exemplos de indicadores de desempenho que tiveram variação mensal média superior a 50%. Dejun, Pierre e Chi (2009) também estudaram a variação de desempenho na Amazon EC2, focando principalmente no desempenho de servidores Web e banco de dados, que chegou a atingir uma variação de 71,1% para um tipo de suas operações. El-Khamra et al. (2010) investigaram o desempenho da Amazon EC2 e FutureGrid com aplicações reais e detectaram muita flutuação nos resultados, com grande parte do tempo de execução sendo gasto com comunicação entre as MVs (chegando a 78% em um dos casos).

A preocupação com a instabilidade do desempenho das nuvens comerciais desencadeou o surgimento de diversos projetos para monitorar o desempenho destas nuvens. Por exemplo, CloudClimate (CLOUDCLIMATE, 2012) e CloudKick[1] (CLOUDKICK, 2012) que realizam monitoramento do desempenho de diferentes nuvens comerciais. A Amazon também fornece uma solução para monitorar os seus serviços e recursos, através do CloudWatch (AMAZON, 2012a).

Com a sobrecarga da virtualização sobre o processamento praticamente zerada, o desempenho de rede é o que mais preocupa os usuários de aplicações MPI e aplicações de uso intensivo de rede. Nesse contexto, Li et al. (2009) propuseram melhorias na implementação da virtualização de rede do hipervisor KVM, melhorando o suporte às aplicações de uso intensivo de rede, especialmente na comunicação entre MVs que compartilham a mesma placa de rede, por estarem na mesma MF. A utilização de *drivers virtio* (RUSSELL, 2008) também mostrou-se eficiente para melhorar o desempenho de rede para MVs sobre o KVM. Com relação ao Xen, o recente trabalho de Pu et al. (2012) lista cinco aspectos que interferem no desempenho de rede e atrapalha provedores de nuvem de serviços e consumidores. Um dos aspectos mais críticos está relacionado à limitação do uso da CPU, e mostra que, apesar do desempenho de rede do Xen ser melhor do que o do KVM, algumas otimizações ainda são necessárias para que aplicações de uso intensivo de rede tenham o mesmo desempenho que em *hardware* real.

6.2 Limitação do uso da CPU e alteração dinâmica de recursos

A limitação da capacidade de processamento já foi explorada em outros trabalhos. Baruchi e Midorikawa (2008) analisam a influência de algoritmos de escalonamento na vazão da rede em um ambiente virtualizado. A influência do algoritmo *Credit Scheduler*, do Xen, sobre o desempenho do I/O da rede é estudada. Lee et al. (2010) propõem um novo escalonador para gerenciar a latência da rede, implementado sobre o Xen, de forma que ele se torne mais adaptável para aplicações em tempo real. Já Bai, Xu e Li (2010) apresentam um escalonador (*co-scheduling*) de CPU orientado a tarefas para um sistema de MVs, baseado em técnicas de inferência, que manipula os núcleos da CPU e a percentagem de CPU alocada para as MVs.

[1] Comprado recentemente pela RackScale.

Este trabalho visa alcançar o máximo de poder de processamento, ao destinar as tarefas para núcleos específicos do processador.

Todos esses trabalhos (BARUCHI; MIDORIKAWA, 2008; LEE et al., 2010; BAI; XU; LI, 2010) utilizaram apenas o Xen, e manipularam o algoritmo padrão de escalonamento, *Credit Scheduler*, para ajustar o uso da CPU pelas MVs, visando diversos objetivos. Até onde foi pesquisado, nenhum outro trabalho procurou ajustar o uso da CPU a fim de garantir que o desempenho da MV, em termos de processamento, seja independente da MF subjacente.

PRESS (GONG; GU; WILKES, 2010) é um sistema de escala preditivo de recursos para computação em nuvem. Ele utiliza técnicas de processamento de sinal e métodos estatísticos para prever as demandas dinâmicas de recursos das aplicações, e prover escalabilidade vertical para as MVs. PRESS altera dinamicamente a quantidade de CPU alocada para a MV, para evitar quebras dos objetivos de nível de serviço (SLOs). Em (SHEN et al., 2011), os autores estenderam a solução anterior para diminuir o consumo de energia e utilizar menos recursos. As duas soluções trabalham apenas com o hipervisor Xen e os autores não consideram a diferença de desempenho causada pela heterogeneidade das MFs.

Sangpetch, Turner e Kim (2010) apresentam um sistema de controle adaptativo, que automatiza a tarefa de ajustar a alocação de recursos e a manutenção dos SLOs. Este trabalho utiliza KVM e a ideia de limitar o uso da CPU, e foca em manter o tempo de resposta esperado para aplicações Web, onde a alocação e o limite de CPU para as MVs são ajustados constantemente através do uso da ferramenta *Control Groups* (*cgroups*). Apesar de haver algumas semelhanças entre as ideias utilizadas neste trabalho e o nosso, principalmente no fato de limitar o uso da CPU e também trabalhar com o KVM, os autores utilizam uma solução diferente para realizar a limitação do uso da CPU e focam em aplicações Web, usando diferentes parâmetros para ajustar o uso da CPU.

Outro trabalho que utilizou o hipervisor KVM e a ferramenta *cpulimit* está em (NIEHOERSTER; BRINKMANN, 2011). Os autores propõem uma solução de *software* localizada na camada de SaaS com agentes autônomos para lidar com as solicitações do usuário. Os agentes ajustam a alocação dos recursos para compensar as flutuações de desempenho. Eles usam uma combinação de máquinas de vetores de suporte (*Support Vector Machines* - SVM) e modelos preditivos para prever e planejar configurações futuras, e assim, evitar quebra de SLOs.

Em um ambiente heterogêneo, pode acontecer de o desempenho ser melhor em uma determinada MF, mesmo utilizando menos CPU, pelo fato de estar numa MF mais potente. Nenhum dos trabalhos citados considera esse fator, pois geralmente aceitam que estão trabalhando em um ambiente homogêneo e que a única causa da variação de desempenho é a dinamicidade das cargas de trabalho. Nosso trabalho, resolve o problema causado pelas MFs heterogêneas e implementa as funcionalidades necessárias para alterar dinamicamente os recursos de processamento, o que é suficiente para implementar qualquer uma das soluções mencionadas acima.

6.3 Escalonamento de máquinas virtuais

Diversos grupos de pesquisa tem utilizado MVs como mecanismos de alocação e gerenciamento de recursos computacionais, tanto para ambientes de grades computacionais, quanto para ambientes de computação em nuvem. Os principais *middleware* de computação em nuvem, Eucalyptus, OpenNebula e Nimbus, vêm com soluções nativas de escalonamento.

O Eucalyptus utiliza algoritmos gulosos do tipo *First Fit* e *Round-robin*. Estas técnicas são métodos aleatórios que utilizam o valor bruto da quantidade de CPU requerida para a MV, e não se preocupam com o desempenho da MV.

O OpenNebula vem com o escalonador *Match-making* que implementa políticas de escalonamento baseada em *Rank*. Este escalonador pode utilizar três políticas de alocação: *Packing Policy*, *Striping Policy* e *Load-aware Policy*. Estas políticas são heurísticas que visam minimizar o número de MFs em uso ou maximizar os recursos disponíveis em uma MF, escolhendo o servidor com mais MVs em execução, com menos MVs em execução ou com mais CPU livre, respectivamente.

O Nimbus usa algumas ferramentas customizadas como *Portable Batch System* (PBS) e *Oracle Grid Engine* (OGE), antigamente chamado *Sun Grid Engine* (SGE), que são bastante utilizados em ambientes de grades computacionais.

Nenhum dos três *middleware* apresenta uma solução de escalonamento que utilize qualquer estratégia para representar e alocar o poder de processamento das MVs, de maneira a garantir o desempenho da MV independente da MF em que ela seja alocada.

Na literatura encontramos diversos trabalhos sobre escalonamento de recursos em sistemas distribuídos. Nota-se que os escalonadores podem ter diversos objetivos, assim como podem ser implementados de diversas formas, baseados em tarefas (*job-based*), máquinas virtuais (*VM-based*), *lease-based*, e outros.

Zhong, Tao e Zhang (2010) propõem um algoritmo genético, baseado em tarefas, para resolver o problema de otimizar o escalonamento e maximizar a utilização dos recursos. Eles usam um algoritmo genético e os menores genes para acelerar a evolução, e introduzem a Política de Dividendos da Economia como Função Objetivo. Porém, eles não consideram o problema de variabilidade no desempenho das MVs, causado pela utilização de MFs heterogêneas.

Stage e Setzer (2009) investigam a alocação de MVs durante a migração. A ideia deles é minimizar o número de servidores em execução, enquanto garantem os SLAs. Eles utilizam um classificador de carga de trabalho para as MVs, em seguida executam um planejador de alocação que leva em consideração o tipo de carga de trabalho para prever as possíveis sobrecargas durante a migração, e por fim, utilizam o escalonador de migração, que tem uma visão geral do *datacenter*, e com base nos planos de migração pode propor algumas realocações antes que ocorra alguma sobrecarga de recursos. Os autores desenvolveram uma abordagem bastante interessante, mas com o foco voltado para os recursos de rede, praticamente desconsiderando o desempenho das MVs, apesar de contarem com as mudanças de carga de trabalho.

Armstrong et al. (2010) apresentam o gerenciador de MVs *Cloud Scheduler*, que visa gerenciar o ciclo de vida das MVs e a submissão de tarefas dos usuários. Eles utilizam o *Condor Job Scheduler* para executar aplicações complexas de larga escala em IaaS.

Fallenbeck et al. (2006) desenvolveram um escalonador, baseado em tarefas, que é uma extensão do SGE e permite alocar tarefas paralelas e seriais no mesmo *cluster*. Os autores basearam o escalonador nas técnicas de virtualização providas pelo hipervisor Xen, que resultou no *Xen Grid Engine* (XGE). Eles utilizam a ideia de se beneficiar das técnicas de *checkpointing*[2], para migrar arbitrariamente tarefas seriais sem precisar fazer modificações nas tarefas.

Outra solução, baseada em tarefas, foi proposta por Chang, Ren e Viswanathan (2010). Os autores definiram um cenário de execução de tarefas em lote, cujo objetivo era minimizar o número ou o custo dos recursos que precisavam ser reservados para que todas as tarefas fossem executadas. Eles formularam um problema de otimização, provaram que era NP-Completo e propuseram um algoritmo aproximado, baseado em técnicas gulosas de aproximação para o problema *Submodular Set Cover* (SSC).

Nenhum destes trabalhos considera as questões relacionadas à variabilidade do desempenho das MVs causada pela heterogeneidade das MFs. Eles trabalham com o valor bruto da quantidade de CPU requerida, sem qualquer preocupação de manter o poder de processamento e reduzir o consumo de energia.

Younge et al. (2010) apresentam um trabalho relacionado à computação verde, onde o objetivo é reduzir ao máximo o consumo de energia, seja fazendo aperfeiçoamentos a nível de *software* e *hardware*, ou otimizando a alocação das MVs. Eles desenvolveram um algoritmo para reduzir o consumo de energia, porém eles consideram um ambiente composto de MFs homogêneas, desconsiderando problemas como a variabilidade de desempenho e o fato de MFs diferentes terem consumo de energia diferentes. Nós estendemos o algoritmo proposto pelos autores para trabalhar com MFs heterogêneos, e alcançamos uma economia de 7,1% no consumo de energia.

Sotomayor et al. (2008), Sotomayor (2010) desenvolveram o Haizea[3] com o conceito de *lease* como uma abstração para o provisionamento de recursos. Os usuários negociam uma quantidade de recursos por um período específico, indicando se a alocação será feita em um modo de melhor esforço ou com reserva antecipada. Dependendo do tipo, os contratos de arrendamento podem ser migrados ou suspensos, para liberar recursos para as locações não-preemptivas. Apesar de ser um dos escalonadores mais completos, esta solução está preocupada com quando as MVs devem ser inicializadas e não consideram mais parâmetros, como o consumo de energia, desempenho das MVs, SLAs ou mudanças de carga de trabalho. O modelo de *leases*, praticamente obriga o usuário a saber de antemão quando a sua carga de trabalho vai aumentar, para que seja possível agendar a inicialização das MVs e realizar a escalabilidade horizontal.

[2]Consiste em salvar o estado atual de um processo, para que ele possa ser reiniciado em caso de falha.

[3]Haizea é uma arquitetura de gerenciamento de *leases* de código aberto, baseadas em MVs, que funciona junto com o OpenNebula.

Como este trabalho propõe uma nova maneira de trabalhar com o poder de processamento das MVs e MFs, é normal não encontrar algoritmos de escalonamento prontos para nosso sistema. Entretanto, apesar do presente trabalho não focar na criação de novas técnicas de escalonamento, foi feito um esforço para adaptar ou aperfeiçoar técnicas da literatura (como as que foram apresentadas na Seção 4.4.3), para que estas possam se adequar à ideia de utilizar a quantidade de UPs requisitadas para a MV, bem como a quantidade de UPs livres nas MFs, no momento de decidir onde as MVs devem ser alocadas.

7 CONCLUSÃO

Computação em nuvem tem o potencial de transformar grande parte da indústria de TI, tornando o *software* ainda mais atraente como um serviço e moldando a forma como o *hardware* é projetado e comprado. O uso de técnicas de virtualização fornece grande flexibilidade com a habilidade de instanciar várias MVs em uma mesma MF, modificar a capacidade das MVs e migrá-las entre as MFs.

A diferença entre as técnicas de virtualização, a heterogeneidade das MFs que compõem a infraestrutura e a dinamicidade da carga de trabalho são as principais causas da variabilidade de desempenho, que é um problema que, comprovadamente, afeta tanto as nuvens privadas, quanto as nuvens públicas.

No presente trabalho, foi proposta uma arquitetura para padronizar a representação do poder de processamento das máquinas físicas e virtuais, em função de unidades de processamento, apoiando-se na limitação do uso da CPU para prover isolamento de desempenho e manter a capacidade de processamento das MVs, independente da MF subjacente.

O trabalho buscou suprir a necessidade de uma solução que considerasse a heterogeneidade das MFs presentes na infraestrutura de uma nuvem computacional. A arquitetura foi descrita e o funcionamento de seus módulos foram apresentados, bem como importantes detalhes de implementação e a estratégia utilizada para definir as UPs em função de GFLOPS, apesar de que outras métricas e estratégias podem ser utilizadas.

Como estudo caso, a arquitetura FairCPU foi implementada para trabalhar com os hipervisores KVM e Xen, e com o *middleware* OpenNebula. Durante o desenvolvimento deste trabalho, foi identificado um problema relacionado ao hipervisor KVM, pela fato dele não ter uma solução nativa para limitar o uso da CPU das MVs. Por isso, a ferramenta *cpulimit* foi avaliada e seu código-fonte modificado para que pudesse ser utilizada em MVs com mais de 1 VCPU.

Diversos experimentos computacionais foram realizados para provar a eficiência da arquitetura FairCPU em manter o poder de processamento das MVs independente da MF subjacente. Aplicações com OpenMP e MPI, além de *benchmarks* de processamento, rede e disco, foram utilizados. Os resultados mostraram que a utilização da FairCPU diminuiu consideravelmente a variação de desempenho da infraestrutura da nossa nuvem privada, e obteve resultados mais constantes do que na Amazon EC2, que é a maior das a nuvens públicas comerciais.

Os resultados mostram também alguns pontos fracos da solução proposta, principalmente no impacto da utilização das UPs sobre o desempenho da rede. Notadamente, a vazão da rede teve uma diminuição de 76,3%, no pior caso, causada pela limitação do uso da CPU da MV. Além disso, a leitura e escrita em disco também sofreu com a perda de desempenho, mais notada no hipervisor Xen. Apesar desses problemas, esperamos que novas versões dos hipervisores KVM e Xen tragam melhorias para reduzir o impacto da limitação da CPU no I/O de rede e disco.

Com a utilização das UPs para representar o poder computacional de MFs e MVs,

surgiu a necessidade de desenvolver políticas de escalonamento que utilizem as UPs, em vez de usar a informação bruta sobre a quantidade de CPU, que mostrou-se propensa a variação de desempenho em ambientes heterogêneos. Para isso, as políticas de escalonamento do *middleware* OpenNebula foram reimplementadas e uma política para redução do consumo de energia foi estendida e aperfeiçoada da literatura.

As principais contribuições do presente trabalho são:

1. Estratégia para definição das unidades de processamento;
2. Definição e implementação da arquitetura FairCPU;
3. Avaliação de ferramenta para limitar o uso da CPU de MVs sobre o hipervisor KVM;
4. Teste de desempenho para MVs executadas sobre os hipervisores Xen e KVM;
5. Implementação de políticas de escalonamento, baseadas em UPs, para a arquitetura FairCPU.

Com essas contribuições, foram realizadas duas publicações: (REGO; COUTINHO; SOUZA, 2011) e (REGO et al., 2011).

Todos os artefatos gerados no presente trabalho estarão disponíveis na Web[1], bem como todas as ferramentas, aplicações e *benchmarks* utilizados, com seus créditos mantidos, seguindo as licenças de uso.

7.1 Trabalhos Futuros

Diversas oportunidades para trabalhos futuros foram apresentadas ao longo do texto. As principais são apresentadas a seguir:

1. Estudar e implementar uma solução para automatizar a definição da relação "limite-poder de processamento", e assim, automatizar e facilitar a definição da UP para diferentes MFs;
2. Estudar mais profundamente o impacto de diferentes cargas de trabalho e diferentes tecnologias no desempenho do processador, para aperfeiçoar a definição das UPs;
3. Desenvolver um sistema adaptativo para automatizar a alteração da quantidade de UPs alocadas para uma MV, adequando o poder computacional às mudanças na carga de trabalho;
4. Estudar o poder de processamento da ECU da Amazon EC2 e definir o mesmo valor para a UP, e assim, trabalhar com nuvens híbridas;
5. Estudar a utilização de *drivers virtio* para melhorar o desempenho do I/O de rede e disco das MVs executadas com o KVM;

[1] `http://sourceforge.net/p/faircpu` e `http://lia.ufc.br/~pauloalr/faircpu`.

6. Utilizar outros *benchmarks* e métricas para ajudar na definição das UPs;
7. Estudar as modificações na versão mais atual do OpenNebula, 3.2, e atualizar a FairCPU para trabalhar com ela;
8. Implementar a FairCPU para funcionar com outros *middleware*;
9. Estudar a utilização da arquitetura FairCPU com outros hipervisores;
10. Implementar novas políticas de escalonamento baseadas em UPs.

REFERÊNCIAS BIBLIOGRÁFICAS

AMAZON. *Amazon CloudWatch*. Jan 2012. Disponível em: <http://aws.amazon.com/cloudwatch>.

AMAZON. *Amazon Web Services*. Jan 2012. Disponível em: <http://aws.amazon.com/>.

ARMBRUST, M. et al. *Above the Clouds: A Berkeley View of Cloud Computing*. [S.l.], Feb 2009. Disponível em: <http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html>.

ARMSTRONG, P. et al. Cloud scheduler: a resource manager for distributed compute clouds. *CoRR*, abs/1007.0050, 2010.

AZURE. *Microsoft Azure*. Jan 2012. Disponível em: <http://www.microsoft.com/azure/>.

BAI, Y.; XU, C.; LI, Z. Task-aware based co-scheduling for virtual machine system. In: *Proceedings of the 2010 ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2010. (SAC '10), p. 181–188. ISBN 978-1-60558-639-7. Disponível em: <http://doi.acm.org/10.1145/1774088.1774126>.

BARUCHI, A.; MIDORIKAWA, E. Influência do algoritmo de escalonamento credit scheduler no desempenho de rede. In: *Workshop de Sistemas Operacionais (WSO), SBC 2008*. [S.l.: s.n.], 2008.

BUYYA, R. et al. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Gener. Comput. Syst.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 25, p. 599–616, June 2009. ISSN 0167-739X. Disponível em: <http://dl.acm.org/citation.cfm?id=1528937.1529211>.

CHAISIRI, S.; LEE, B.-S.; NIYATO, D. Optimal virtual machine placement across multiple cloud providers. In: *Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific*. Singapore: [s.n.], 2009.

CHANG, F.; REN, J.; VISWANATHAN, R. Optimal resource allocation in clouds. In: *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. Miami, FL: [s.n.], 2010.

CHEN, P. M.; NOBLE, B. D. When virtual is better than real [operating system relocation to virtual machines]. In: *Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on*. [S.l.: s.n.], 2001.

CHERKASOVA, L.; GUPTA, D.; VAHDAT, A. Comparison of the three cpu schedulers in xen. *SIGMETRICS Perform. Eval. Rev.*, ACM, New York, NY, USA, v. 35, p. 42–51, September 2007. ISSN 0163-5999. Disponível em: <http://doi.acm.org/10.1145/1330555.1330556>.

CIURANA, E. *Developing with Google App Engine*. Berkely, CA, USA: Apress, 2009. ISBN 1430218312, 9781430218319.

CLOUDCLIMATE. Jan 2012. Disponível em: <http://www.cloudclimate.com>.

CLOUDKICK. Jan 2012. Disponível em: <https://www.cloudkick.com>.

CPULIMIT. *CPU Usage Limiter for Linux*. Jan 2012. Disponível em: <http://cpulimit.sourceforge.net/>.

CREDITSCHEDULER. *Credit-Based CPU Scheduler*. Jan 2012. Disponível em: <http://wiki.xensource.com/xenwiki/CreditScheduler>.

DEJUN, J.; PIERRE, G.; CHI, C.-H. Ec2 performance analysis for resource provisioning of service-oriented applications. In: *Proceedings of the 2009 international conference on Service-oriented computing*. Berlin, Heidelberg: Springer-Verlag, 2009. (ICSOC/ServiceWave'09), p. 197–207. ISBN 3-642-16131-6, 978-3-642-16131-5. Disponível em: <http://dl.acm.org/citation.cfm?id=1926618.1926641>.

DESHANE, T. et al. Quantitative comparison of Xen and KVM. In: *Xen summit*. Berkeley, CA, USA: USENIX association, 2008.

DORTA, A.; RODRIGUEZ, C.; SANDE, F. de. The openmp source code repository. In: *Parallel, Distributed and Network-Based Processing, 2005. PDP 2005. 13th Euromicro Conference on*. [S.l.: s.n.], 2005. p. 244 – 250. ISSN 1066-6192.

EL-KHAMRA, Y. et al. Exploring the performance fluctuations of hpc workloads on clouds. In: *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*. [S.l.: s.n.], 2010. p. 383 –387.

ENDO, P. T. et al. A survey on open-source cloud computing solutions. In: *VIII Workshop em Clouds, Grids e Aplicações. Simpósio Brasileiro de Redes de Computadores e de Sistemas Distribuídos*. [S.l.: s.n.], 2010.

EUCALYPTUS. *Eucalyptus Open Source Cloud Platform*. Jan 2012. Disponível em: <http://open.eucalyptus.com/>.

EVOY, G. V. M.; SCHULZE, B.; GARCIA, E. L. M. Performance and deployment evaluation of a parallel application on a private cloud. *Concurrency and Computation: Practice and Experience*, John Wiley e Sons, Ltd., p. n/a–n/a, 2011. ISSN 1532-0634. Disponível em: <http://dx.doi.org/10.1002/cpe.1699>.

FALLENBECK, N. et al. Xen and the art of cluster scheduling. In: *Virtualization Technology in Distributed Computing, 2006. VTDC 2006. First International Workshop on*. Tampa, FL: [s.n.], 2006.

GALANTE, G.; BONA, L. C. E. Nebulous: A framework for scientific applications execution on cloud environments. In: *Anais do XII Simp. em Sistemas Computacionais de Alto Desempenho*. [S.l.: s.n.], 2011. (WSCAD 2011).

GOGRID. *GoGrid Cloud Solutions*. Jan 2012. Disponível em: <http://www.gogrid.com/>.

GONG, Z.; GU, X.; WILKES, J. Press: Predictive elastic resource scaling for cloud systems. In: *Network and Service Management (CNSM), 2010 International Conference on*. [S.l.: s.n.], 2010. p. 9 –16.

GRIT, L. et al. Virtual machine hosting for networked clusters: Building the foundations for "autonomic"orchestration. In: *Virtualization Technology in Distributed Computing, 2006. VTDC 2006. First International Workshop on*. Tampa, FL: [s.n.], 2006.

HENDERSON, R. Job scheduling under the portable batch system. In: FEITELSON, D.; RUDOLPH, L. (Ed.). *Job Scheduling Strategies for Parallel Processing*. [S.l.]: Springer Berlin / Heidelberg, 1995, (Lecture Notes in Computer Science, v. 949). p. 279–294.

HUU, T. T.; MONTAGNAT, J. Virtual resources allocation for workflow-based applications distribution on a cloud infrastructure. In: *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. Washington, DC, USA: IEEE Computer Society, 2010. (CCGRID '10), p. 612–617. ISBN 978-0-7695-4039-9. Disponível em: <http://dx.doi.org/10.1109/CCGRID.2010.23>.

IBM. *Virtualization in Education*. Oct 2007. White paper, IBM Systems and Technology Group. Disponível em: <http://www-07.ibm.com/solutions/in/education/download/Virtualization in Education.pdf>.

INTEL. *Intel Optimized LINPACK Benchmark*. Jan 2012. Disponível em: <http://software.intel.com/en-us/articles/intel-math-kernel-library-linpack-download/>.

IOSUP, A. et al. Performance analysis of cloud computing services for many-tasks scientific computing. *IEEE Transactions on Parallel and Distributed Systems*, IEEE Computer Society, Los Alamitos, CA, USA, v. 22, p. 931–945, 2011. ISSN 1045-9219.

IOSUP, A.; YIGITBASI, N.; EPEMA, D. On the performance variability of production cloud services. In: *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*. [S.l.: s.n.], 2011. p. 104–113.

IOZONE. *IOzone Filesystem Benchmark*. Jan 2012. Disponível em: <http://www.iozone.org/>.

KEAHEY, K. et al. Virtual workspaces for scientific applications. *Journal of Physics: Conference Series*, v. 78, n. 1, p. 012038, 2007. Disponível em: <http://stacks.iop.org/1742-6596/78/i=1/a=012038>.

KHATUA, S.; GHOSH, A.; MUKHERJEE, N. Optimizing the utilization of virtual resources in cloud environment. In: *Virtual Environments Human-Computer Interfaces and Measurement Systems (VECIMS), 2010 IEEE International Conference on*. Taranto: [s.n.], 2010. ISSN 1944-9429.

KRISHNAN, B. et al. Vm power metering: feasibility and challenges. *SIGMETRICS Perform. Eval. Rev.*, ACM, New York, NY, USA, v. 38, p. 56–60, January 2011. ISSN 0163-5999. Disponível em: <http://doi.acm.org/10.1145/1925019.1925031>.

LEE, M. et al. Supporting soft real-time tasks in the xen hypervisor. In: *Proceedings of the 6th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. New York, NY, USA: ACM, 2010. (VEE '10), p. 97–108. ISBN 978-1-60558-910-7. Disponível em: <http://doi.acm.org/10.1145/1735997.1736012>.

LI, S. et al. Optimizing network virtualization in kernel-based virtual machine. In: *Proceedings of the 2009 First IEEE International Conference on Information Science and Engineering*. Washington, DC, USA: IEEE Computer Society, 2009. (ICISE '09), p. 282–285. ISBN 978-0-7695-3887-7. Disponível em: <http://dx.doi.org/10.1109/ICISE.2009.813>.

MELL, P.; GRANCE, T. The nist definition of cloud computing. *National Institute of Standards and Technology*, NIST, v. 53, n. 6, p. 50, 2009. Disponível em: <http://csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc>.

NAPPER, J.; BIENTINESI, P. Can cloud computing reach the top500? In: *Proceedings of the combined workshops on UnConventional high performance computing workshop plus memory access workshop*. New York, NY, USA: ACM, 2009. (UCHPC-MAW '09), p. 17–20. ISBN 978-1-60558-557-4.

NIEHOERSTER, O.; BRINKMANN, A. Autonomic resource management handling delayed configuration effects. In: *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*. [S.l.: s.n.], 2011. p. 138 –145.

OPENNEBULA. *The Open Source Toolkit for Cloud Computing*. Jan 2012. Disponível em: <http://opennebula.org/>.

OSTERMANN, S. et al. A Performance Analysis of EC2 Cloud Computing Services for Scientific Computing. In: *Cloud Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, (Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, v. 34). cap. 9, p. 115–131. ISBN 978-3-642-12635-2. Disponível em: <http://dx.doi.org/10.1007/978-3-642-12636-9_9>.

PU, X. et al. Who is your neighbor: Net i/o performance interference in virtualized clouds. *IEEE Transactions on Services Computing*, IEEE Computer Society, Los Alamitos, CA, USA, v. 99, n. PrePrints, 2012. ISSN 1939-1374.

RACKSPACE. *RackSpace Hosting*. Jan 2012. Disponível em: <http://www.rackspace.com/>.

REGO, P. A. L. et al. Faircpu: Architecture for allocation of virtual machines using processing features. In: *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*. [S.l.: s.n.], 2011. p. 371 –376.

REGO, P. A. L.; COUTINHO, E. F.; SOUZA, J. N. de. Proposta de workflow para alocação de máquinas virtuais utilizando características de processamento. In: *IX Workshop em Clouds, Grids e Aplicações. Simpósio Brasileiro de Redes de Computadores e de Sistemas Distribuídos*. [S.l.: s.n.], 2011.

REGOLA, N.; DUCOM, J.-C. Recommendations for virtualization technologies in high performance computing. In: *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*. [S.l.: s.n.], 2010. p. 409 –416.

RIMAL, B. P.; CHOI, E.; LUMB, I. A taxonomy and survey of cloud computing systems. In: *INC, IMS and IDC, 2009. NCM '09. Fifth International Joint Conference on*. Seoul: [s.n.], 2009.

RUSSELL, R. virtio: towards a de-facto standard for virtual i/o devices. *SIGOPS Oper. Syst. Rev.*, ACM, New York, NY, USA, v. 42, p. 95–103, July 2008. ISSN 0163-5980. Disponível em: <http://doi.acm.org/10.1145/1400097.1400108>.

SALESFORCE. *Salesforce CRM*. Jan 2012. Disponível em: <http://www.salesforce.com/>.

SANGPETCH, A.; TURNER, A.; KIM, H. How to tame your vms: an automated control system for virtualized services. In: *Proceedings of the 24th international conference on Large installation system administration*. Berkeley, CA, USA: USENIX Association, 2010. (LISA'10), p. 1–16. Disponível em: <http://portal.acm.org/citation.cfm?id=1924976.1924995>.

SCHAD, J.; DITTRICH, J.; QUIANÉ-RUIZ, J.-A. Runtime measurements in the cloud: observing, analyzing, and reducing variance. *Proc. VLDB Endow.*, VLDB Endowment, v. 3, p. 460–471, September 2010. ISSN 2150-8097.

SEMPOLINSKI, P.; THAIN, D. A comparison and critique of eucalyptus, opennebula and nimbus. In: *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*. Indianapolis, IN: [s.n.], 2010.

SHEN, Z. et al. Cloudscale: elastic resource scaling for multi-tenant cloud systems. In: *Proceedings of the 2nd ACM Symposium on Cloud Computing*. New York, NY, USA: ACM, 2011. (SOCC '11), p. 5:1–5:14. ISBN 978-1-4503-0976-9. Disponível em: <http://doi.acm.org/10.1145/2038916.2038921>.

SMITH, J. E.; NAIR, R. The architecture of virtual machines. *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 38, p. 32–38, May 2005. ISSN 0018-9162. Disponível em: <http://portal.acm.org/citation.cfm?id=1069588.1069632>.

SONNEK, J.; CHANDRA, A. Virtual Putty: Reshaping the Physical Footprint of Virtual Machines. In: *Proc. of Workshop on Hot Topics in Cloud Computing (HotCloud'09)*. [S.l.: s.n.], 2009.

SOROR, A. A. et al. Automatic virtual machine configuration for database workloads. *ACM Trans. Database Syst.*, ACM, New York, NY, USA, v. 35, p. 7:1–7:47, February 2010.

SOTOMAYOR, B. *Provisioning computational resources using virtual machines and leases*. Tese (Doutorado) — The University of Chicago, Illinois, USA, 2010.

SOTOMAYOR, B. et al. Capacity Leasing in Cloud Systems using the OpenNebula Engine. In: CLOUD COMPUTING AND APPLICATIONS 2008 (CCA08). [S.l.], 2008.

SOUSA, F. R. C. et al. Gerenciamento de dados em nuvem: Conceitos, sistemas e desafio. In: *Minicursos. Simpósio Brasileiro de Banco de Dados*. [S.l.: s.n.], 2011.

SRINIVASAN, S. et al. Characterization of backfilling strategies for parallel job scheduling. In: *Parallel Processing Workshops, 2002. Proceedings. International Conference on*. [S.l.: s.n.], 2002. p. 514 – 519. ISSN 1530-2016.

STAGE, A.; SETZER, T. Network-aware migration control and scheduling of differentiated virtual machine workloads. In: *Software Engineering Challenges of Cloud Computing, 2009. CLOUD '09. ICSE Workshop on*. Vancouver, BC: [s.n.], 2009.

TANNENBAUM, T. et al. Beowulf cluster computing with linux. In: . Cambridge, MA, USA: MIT Press, 2002. cap. Condor: a distributed job scheduler, p. 307–350.

TIRUMALA, A.; COTTRELL, L.; DUNIGAN, T. Measuring end-to-end bandwidth with iperf using web100. In: *Web100", Proc. of Passive and Active Measurement Workshop*. [S.l.: s.n.], 2003. p. 2003.

TOP500.ORG. *Top 500 Supercomputer Sites*. Jan 2012. Disponível em: <http://www.top500.org/>.

VAQUERO, L. M. et al. A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 39, p. 50–55, December 2008. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1496091.1496100>.

VECCHIOLA, C.; PANDEY, S.; BUYYA, R. High-performance cloud computing: A view of scientific applications. In: *Proceedings of the 2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks*. Washington, DC, USA: IEEE Computer Society, 2009. (ISPAN '09), p. 4–16. ISBN 978-0-7695-3908-9. Disponível em: <http://dx.doi.org/10.1109/I-SPAN.2009.150>.

VERDI, F. L. et al. Novas arquiteturas de data center para cloud computing. In: *Minicursos. Simpósio Brasileiro de Redes de Computadores e de Sistemas Distribuídos*. [S.l.: s.n.], 2010.

VMWARE. *Understanding Full Virtualization, Paravirtualization, and Hardware Assist*. Sept 2007. White paper, VMware Inc.

VMWARE. *The Benefits of Virtualization for Small and Medium Businesses*. May 2011. White paper, VMware Inc. Disponível em: <http://www.vmware.com/files/pdf/VMware-SMB-Survey.pdf>.

VOUK, M. Cloud computing - issues, research and implementations. In: *Information Technology Interfaces, 2008. ITI 2008. 30th International Conference on*. [S.l.: s.n.], 2008. p. 31 –40. ISSN 1330-1012.

WANG, L. et al. Cloud computing: a perspective study. *New Generation Computing*, Ohmsha, Ltd., v. 28, p. 137–146, 2010. ISSN 0288-3635. 10.1007/s00354-008-0081-5. Disponível em: <http://dx.doi.org/10.1007/s00354-008-0081-5>.

YOUNGE, A. J. et al. Efficient resource management for cloud computing environments. In: *Proceedings of the International Conference on Green Computing*. Washington, DC, USA: IEEE Computer Society, 2010. (GREENCOMP '10), p. 357–364. ISBN 978-1-4244-7612-1. Disponível em: <http://dx.doi.org/10.1109/GREENCOMP.2010.5598294>.

ZHANG, Q.; CHENG, L.; BOUTABA, R. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, Springer London, v. 1, n. 1, p. 7–18, maio 2010. ISSN 1867-4828. Disponível em: <http://dx.doi.org/10.1007/s13174-010-0007-6>.

ZHONG, H.; TAO, K.; ZHANG, X. An approach to optimized resource scheduling algorithm for open-source cloud systems. In: *ChinaGrid Conference (ChinaGrid), 2010 Fifth Annual*. Guangzhou: [s.n.], 2010.

ZHOU, S. Virtual networking. *SIGOPS Oper. Syst. Rev.*, ACM, New York, NY, USA, v. 44, p. 80–85, dez. 2010. ISSN 0163-5980. Disponível em: <http://doi.acm.org/10.1145/1899928.1899938>.