Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

Just in Time Clouds: Uma Abordagem Baseada em Recursos Terceirizados para a Ampliação da Elasticidade de Provedores de Computação na Nuvem

Rostand Edson Oliveira Costa

Tese submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Doutor em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Metodologia e Técnicas da Computação

Francisco Vilar Brasileiro
(Orientador)

Campina Grande, Paraíba, Brasil

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

C837j Costa, Rostand Edson Oliveira.

Just in time clouds : uma abordagem baseada em recursos terceirizados para a ampliação da elasticidade de provedores de computação na nuvem / Rostand Edson Oliveira Costa. – Campina Grande, 2013.

172 f. : il. color.

Tese (Doutorado em Ciência da Computação) - Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, 2013.

"Orientação: Prof. Dr. Francisco Vilar Brasileiro".

Referências.

1. Computação na Nuvem. 2. Elasticidade. 3. Federação de Recursos. 4. Recursos Terceirizados. I. Brasileiro, Francisco Vilar. II. Título.

CDU 004.7(043)

"JUST IN TIME CLOUDS: UMA ABORDAGEM BASEADA EM RECURSOS TERCEIRIZADOS PARA A AMPLIAÇÃO DA ELASTICIDADE DE PROVEDORES DE COMPUTAÇÃO NA NUVEM"

ROSTAND EDSON OLIVEIRA COSTA

TESE APROVADA EM 05/03/2013

FRANCISCO VILAR BRASILEIRO, Ph.D, UFCG
Orientador(a)

JOSÉ ANTÃO BELTRÃO MOURA, Ph.D, UFCG
Examinador(a)

JACQUES PHILIPPE SAUVÉ, Ph.D, UFCG
Examinador(a)

PHILIPPE OLIVIER ALEXANDRE NAVAUX, Dr., UFRS
Examinador(a)

CARLOS ANDRE GUIMARAES FERRAZ, Ph.D, UFPE
Examinador(a)

CAMPINA GRANDE - PB

Resumo

A vazão obtida quando se executam aplicações HTC (do inglês *High Throughput Computing*) sobre uma infraestrutura computacional depende diretamente da escala que a mesma permite. Neste contexto, o tamanho do *pool* de processamento é o principal promotor de desempenho, enquanto que o esforço de coordenação envolvido é o principal fator de limitação.

O paradigma da computação na nuvem permite o fornecimento de infraestrutura de Tecnologia da Informação sob a forma de um serviço que os clientes adquirem sob demanda e pagam apenas pela quantidade de serviços que realmente consomem. Muitas aplicações que processam grandes cargas de trabalho em paralelo poderiam potencialmente se beneficiar da elasticidade oferecida pelos provedores de computação na nuvem. Infelizmente, os provedores públicos atuais de computação na nuvem precisam impor um limite estrito na quantidade de recursos que um único usuário pode adquirir concomitantemente.

Para lidar com tal limitação, nós apresentamos uma abordagem alternativa para a construção de infraestruturas computacionais para suporte à computação na nuvem que não é baseada em planejamento de capacidade tradicional. Inspirados na filosofia *Just in Time* (JiT) da Toyota, nós introduzimos o conceito de *Just in Time Clouds* para representar uma nova categoria de serviço na qual o provedor apenas obtém recursos para alocação quando efetivamente demandado pelos clientes e somente enquanto houver uso para eles.

Explorando recursos terceirizados de baixa escala, um fornecedor de uma *JiT Cloud* pode aumentar a sua capacidade de oferecer IaaS de uma forma mais escalável e com uma elasticidade virtualmente ilimitada, uma vez que é baseada na descoberta, federação e revenda de recursos ociosos cujos custos de montagem e operação são pagos por terceiros.

Foi realizada uma prova de conceito usando uma rede de TV Digital para averiguar o potencial de utilização de recursos terceirizados de alta granularidade, alta volatilidade e alta dispersão para a construção de *JiT Clouds* de alta vazão usando uma arquitetura nova: *On-demand Distributed Computing Infrastructure* (OddCI).

Os nossos resultados mostram que é possível montar infraestruturas computacionais dinâmicas baseadas em recursos computacionais posicionados em praticamente todo o espectro de recursos terceirizados de baixa escala. Nos cenários mais desafiadores, foi possível obter disponibilidade coletiva de dispositivos isolados para entregar vazão computacional com perdas máximas de 10% sob regimes de até 40% de volatilidade, causada por falhas ou abandonos voluntários de nós.

Considerando o uso de recursos terceirizados não convencionais, como receptores de TV Digital de baixo custo, foi observada uma diferença relevante de capacidade computacional quando comparados com dispositivos convencionais, mesmo os de baixa granularidade, como PCs domésticos. Entretanto, essa perda não se constitui em uma limitação técnica irreparável mas, tão somente, um aspecto mercadológico e circunstancial, passível de ser contornado com facilidade caso uma demanda para dispositivos mais potentes seja criada.

Palavras-chave: Elasticidade, Computação na Nuvem, Federação de Recursos e Recursos Terceirizados.

Abstract

The throughput obtained when executing HTC (*High Throughput Computing*) applications on a computing infrastructure depends directly on the scale that it offers. In this context, the size of the processing pool is the principal promoter of performance, while the coordination effort involved is the main limiting factor.

The paradigm of cloud computing enables the delivery of Information Technology infrastructure in the form of a service that customers purchase on-demand and pay only for the amount of services that they actually consume. Many applications that process large workloads in parallel could potentially benefit from the elasticity offered by cloud computing providers. Unfortunately, current public cloud computing providers need to impose a strict limit on the amount of resources that a single user can simultaneously acquire.

To address this limitation, we present an alternative approach to the construction of computational infrastructures to support cloud computing that is not based on traditional capacity planning. Inspired by Toyota's *Just in Time* (JiT) philosophy, we introduce the concept of *Just in Time Clouds* to represent a new category of service in which the provider allocates resources only when actually demanded by customers and only while there is use for them.

Exploring low scale outsourced resources, a JiT Cloud provider can increase its ability to offer IaaS in a more scalable way and with a virtually unlimited elasticity, since it is based on the discovery, federation and reselling of idle resources whose installation and operation costs are paid by a third party.

We performed a proof of concept, on a network of Digital TV, to investigate the potential of utilization of outsourced resources with high granularity, high volatility and high dispersion for the construction of *JiT Clouds* with high throughput using a new architecture, called *On-demand Distributed Computing Infrastructure* (OddCI).

Our results show that it is possible to build dynamic computing infrastructures based on computational resources placed in virtually the entire spectrum of low scale outsourced resources. In the most challenging scenarios, it was possible to obtain collective availability using isolated devices to deliver computational throughput with maximum losses of 10% under scenarios of up to 40% of volatility, caused by node unavailability.

Considering the use of unconventional outsourced resources, as low cost Digital TV receivers , there was a significant difference in computational power compared with conventional low granularity devices, such as home PCs. However, this loss does not constitute an irreparable technical limitation, but only one circumstantial marketing aspect, that can be easily circumvented if a demand for more powerful devices is created.

Keywords: Elasticity, Cloud Computing, Resource Federation and Outsourced Resources.

Dedicatória

Dedico este trabalho aos meus pais, Acácio Costa e Carmita Costa, cujo exemplo é fonte de inspiração para todos a sua volta, e aos meus filhos, Giulia e Renan, para quem eu espero poder transmitir, tão fortemente, os mesmos valores e princípios com os quais fui educado.

Agradecimentos

Agradeço a todos os meus familiares e amigos que tanto me incentivaram a prosseguir com este projeto. Em particular, agradeço a Gilvandro, Dr. Vicente, Geórgia, Helga e Jacques, por me proporcionarem, de maneira própria e nos momentos apropriados, os recursos que eu precisava para seguir em frente.

Agradeço às equipes do LSD/UFCG e do LAVID/UFPB pela acolhida e pelo inestimável suporte logístico e técnico. Em especial, gostaria de destacar a relevante participação dos professores Guido Lemos e Dênio Mariz durante toda a condução desta pesquisa.

Agradeço ao meu orientador, Francisco Brasileiro (Fubica), pela generosidade em compartilhar a sua experiência, por todo e tempo e energia que empregou neste trabalho e, principalmente, por ter aceito me acompanhar nesta jornada.

Finalmente, agradeço a minha melhor metade, Gilka, por sua paciência e companheirismo neste e em todos os momentos que passamos juntos.

Conteúdo

Lista de Símbolos

ABNT - *Associação Brasileira de Normas Técnicas*

ACAP - *Advanced Common Application Platform*

AIT - *Application Information Table*

API - *Application Program Interface*

ARIB - *Association of Radio Industries and Businesses* ATSC - *Advanced Television Systems Committee*

AWS - *Amazon Web Services*

BLAST - *Basic Local Alignment Search Tool*

BoT - *Bag-of-Tasks*

CAPEX - *Capital Expenditure*

CRM - *Customer Relationship Management*

DC - *Data Center*

DCI - *Distributed Computing Infrastructures*

DoE - *Design of Experiment*

DSM-CC - *Digital Storage Media Command and Control* DTV - *Digital Television*

DVB - *Digital Video Broadcasting*

DVE - *Dynamic Virtual Environment*

EaaS - *Everything-as-a-Service*

EC2 - *Elastic Compute Cloud*

EP - *Energy Proportionality*

ERB - *Estação Rádio Base*

ETSI - *European Telecommunications Standards Institute*

GEM - *Globally Executable MHP)*

HPC - *High Performance Computing*

HTC - *High Throughput Computing*

IaaS - *Infrastructure-as-a-Service*

IEC - *International Electrotechnical Commission*

ISDB - *Integrated Services Digital Broadcasting*

ISO - *International Organization for Standardization*

ITU - *International Telecommunication Union*

JiT - *Just in Time*

LAVID - *Laboratório de Aplicações de Vídeo Digital*

MHP - *Multimedia Home Platform*

MPEG - *Moving Picture Experts Group*

MTC - *Many Task Computing*

NCBI - *U.S. National Center for Biotechnology Information*

NCL - *Nested Context Language*

OddCI - *On-Demand Distributed Computing Infrastructures*

OPEX - *Operational Expenditure*

OVF - *Open Virtualized Format*

PaaS - *Platform-as-a-Service*

PC - *Personal Computer*

PID - *Packet Identification*

PMT - *Program Map Table*

PNA - *Processing Node Agent*

PUE - *Power Usage Efficiency*

QAM - *Quadrature Amplitude Modulation*

QoS - *Quality of Service*

RDP - *Remote Desktop Protocol*

RFB - *Remote Framebuffer Protocol*

RM - *Reset Message*

SaaS - *Software-as-a-Service*

SAN - *Stochastic Activity Network*

SBTVD - *Sistema Brasileiro de TV Digital*

SI - *Service Information*

SLA - *Service Level Agreement*

SSH - *Secure Shell*

STB - *Set-Top-Box*

TCO - *Total Cost of Ownership*

TI - *Tecnologia da Informação*

TPS - *Toyota Production System*

TS - *Transport Stream*

TVDI - *Televisão Digital Interativa*

UC - *Utilization Cost*

UC - *Uninterrupted Power Supply*

VM - *Virtual Machine*

VPN - *Virtual Private Network*

WM - *Wakeup Message*

WP - *Wakeup Process*

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Computação na nuvem (do inglês *cloud computing*) é um paradigma em evolução que permite o fornecimento de Tecnologia da Informação (TI) como um serviço que pode ser adquirido interativamente, *on line* e sob demanda pelos clientes. Os recursos utilizados para prover serviço aos clientes podem ser rapidamente provisionados e liberados pelos provedores do serviço. Quando o serviço é cobrado dos clientes, os provedores utilizam um modelo de tarifação onde o cliente paga apenas pelo que foi efetivamente consumido. Este paradigma pode ser usado em diferentes níveis da pilha de TI [Stanoevska-Slabeva e Wozniak 2010]. Por exemplo, no nível mais alto, clientes podem adquirir serviços que provêem uma funcionalidade particular de *software*. Este tipo de fornecimento de TI é normalmente chamado de SaaS (do inglês, *software-as-a-service*) [Stanoevska-Slabeva e Wozniak 2010]. Por outro lado, no nível mais baixo da pilha, clientes podem adquirir máquinas virtuais totalmente funcionais executando um determinado sistema operacional, sobre o qual eles podem instalar e executar as suas próprias aplicações. Este tipo de serviço recebeu o nome de IaaS (do inglês, *infrastructure-as-a-service*) [Stanoevska-Slabeva e Wozniak 2010] e é nele que este trabalho está focado[1].

Ao adquirir recursos de TI de um provedor de computação na nuvem, os clientes podem desfrutar da elasticidade oferecida, podendo aumentar e diminuir o seu consumo de serviços de uma forma virtualmente ilimitada, sem qualquer custo adicional. Em teoria, essa elasticidade ilimitada permitiria aos usuários decidirem livremente, por exemplo, se desejam usar 1

[1]No restante deste documento, os termos computação na nuvem e IaaS serão usados de forma intercambiável e com o mesmo propósito.

recurso por 1.000 horas ou 1.000 recursos por 1 hora, pagando o mesmo preço em ambos os casos. Essa propriedade singular de computação na nuvem é chamada de *associatividade de custos* (*cost associativity*) [Fox 2011].

Ao traduzir infraestrutura de TI em serviços elásticos e ilimitados, utilizados sob demanda e pagos de acordo com a quantidade de serviço consumida, o paradigma de computação na nuvem oferece inúmeras possibilidades novas para o planejamento de capacidade das instituições que utilizam TI de forma intensiva. Em particular, a capacidade de instanciar concomitantemente um grande número de recursos por um período de tempo relativamente curto é um requisito fundamental para um modelo de programação de aplicações paralelas cada vez mais popular, chamado computação de alta vazão (HTC, do inglês *High-Throughput Computing*) [Litzkow, Livny e Mutka 1988]. Essas aplicações têm cargas de trabalho altamente paralelizáveis e quanto mais cedo a sua execução possa ser concluída, melhor. Assim, idealmente, elas poderiam ser executadas simultaneamente pela totalidade dos recursos necessários para terminar o mais rapidamente possível e, ainda, com um custo que só dependeria da carga de trabalho que tiver sido realmente processada. Desta forma, muitas aplicações HTC, científicas ou comerciais, poderiam potencialmente obter um enorme benefício a partir da elasticidade dos fornecedores de computação em nuvem.

Infelizmente, os provedores públicos atuais de IaaS precisam limitar o número máximo de instâncias que podem ser adquiridas simultaneamente por um dado cliente e permitem somente que poucas máquinas virtuais sejam instanciadas automatica e concomitantemente pelo mesmo cliente. Por exemplo, durante todo o tempo de desenvolvimento desta pesquisa, o serviço EC2 (*Elastic Compute Cloud*) da *Amazon Web Services* (AWS), um dos principais provedores comerciais em atividade, limitava em 20 o número de máquinas virtuais que podem ser instanciadas de forma dedicada (*on-demand instances*) e em 100 o número de máquinas virtuais que podem ser instanciadas segundo um modelo *"best-effort"* (*spot instances*) [Amazon 2011]. Para este provedor em particular, clientes podem usar um canal paralelo de negociação para tentar aumentar este limite de forma *ad hoc*, mas como as condições sob as quais uma negociação é bem sucedida não são documentadas, nós consideramos neste trabalho apenas o canal de comunicação automático.

Embora os limites atualmente impostos pelos provedores de IaaS não impeçam que a maioria dos clientes enxerguem o serviço provido como uma fonte infinita de recursos,

este não é o caso para a maioria das aplicações HTC. Estas aplicações podem requerer a instanciação de um sistema com milhares de máquinas virtuais. Além disso, quanto mais máquinas elas puderem usar, mais curto será o tempo de utilização das mesmas. O projeto Belle II Monte Carlo [Sevior, Fifield e Katayama 2010], por exemplo, requer de 20.000 a 120.000 máquinas virtuais para o processamento, em tempo aceitável, dos dados produzidos em três meses de experimentos. Ou seja, eles têm uma altíssima demanda por recursos de forma bastante esporádica. Esse padrão de consumo é muito comum entre os usuários que executam aplicações HTC e, possivelmente, também para outras classes de aplicações.

Como já existem serviços de alta demanda hospedados em provedores de IaaS públicos e privados (ex. Gmail, Twitter, Bing etc.) e também a possibilidade de se negociar alocações superiores com provedores públicos, é possível inferir que o limite serve como um regulador do uso intensivo de recursos por períodos curtos, ou seja, o alvo do limite não é o volume da requisição em si, mas o exercício extremo da elasticidade através de grandes alocações com liberações logo em seguida. Desta forma, embora as infraestruturas de computação em nuvem sejam muito flexíveis e fáceis de configurar, não é fácil atingir computação de vazão extremamente alta nelas, considerando as implementações disponíveis.

A baixa amplitude da elasticidade dos provedores atuais de nuvens reflete duas realidades diferentes. Da perspectiva do cliente, o modelo de computação em nuvem permite que este aplique aos seus investimentos em TI os mesmos princípios do *Toyota Production System* (TPS) [Toyota Motor Co 2011]. Criada pela Toyota nos anos 50, a filosofia de sistema de produção "Just in Time" (JiT) é baseada em uma idéia muito simples: "o que é necessário, quando necessário e na quantidade necessária". Os provedores de IaaS, por sua vez, não possuem as mesmas facilidades quando estão montando a infraestrutura sobre as quais eles irão prover os seus serviços, tendo que lidar com a complexidade e riscos associados com o planejamento de capacidade de longa duração.

Para lidar com esta limitação e como contribuição principal desta pesquisa, nós propomos o conceito de *Just in Time Clouds* (*JiT Clouds*) [Costa et al. 2011f], uma abordagem na qual os provedores de serviço apenas incorrem em custos de provisionamento quando os recursos que eles usam para fornecer os seus serviços são demandados pelos seus clientes e apenas durante o período que eles são necessários. Isto alivia os riscos e custos do planejamento de capacidade envolvidos tanto com sub-provisionamento quanto com excesso de

provisionamento de recursos. Para tal, provedores de *JiT Clouds* utilizam apenas o poder de processamento ocioso de recursos pertencentes a terceiros.

Do ponto de vista da escala, os detentores de recursos computacionais ociosos considerados aqui podem ser classificados em duas categorias principais: a) os que possuem capacidade excedente suficiente para poderem atuar como provedores públicos de IaaS, oferecendo os seus recursos ociosos diretamente para os usuários, como fez a Amazon Bookstore, por exemplo, dando origem à AWS; e b) os que não possuem, sozinhos, recursos ociosos suficientes para uma atuação solo no mercado de IaaS.

A última categoria, que chamamos de *recursos terceirizados de pequena escala*, envolve todo o espectro de escala imediatamente inferior ao nível esperado para a primeira categoria, incluindo desde as empresas de grande porte, passando por *data centers* de pequeno porte e chegando até servidores e recursos individuais, convencionais ou não convencionais, pertencentes a instituições ou a indivíduos. Explorando tais recursos terceirizados ociosos, um fornecedor de *JiT Cloud* pode aumentar a sua capacidade de oferecer IaaS de uma forma mais escalável e com uma elasticidade virtualmente ilimitada, uma vez que é baseada na descoberta, federação e revenda de recursos ociosos cujos custos de montagem e operação são pagos por terceiros.

No restante deste capítulo, nós discutimos a relevância deste trabalho (Seção 1.1), apresentamos as suas principais contribuições (Seção 1.2) e delineamos a organização do restante do documento (Seção 1.3).

1.1 Justificativa e Relevância

A comunidade científica não está indiferente ao fenômeno da computação na nuvem e inúmeras iniciativas em todo o mundo já investigam a aplicabilidade do novo ambiente para computação científica ou e-ciência (do inglês *e-science*) [Evangelinos e Hill 2008; Juve et al. 2009; Keahey 2010; Oliveira, Baião e Mattoso 2011; Iosup et al. 2008; Walker 2008]. É reconhecido que muitos dos avanços recentes em pesquisas científicas somente foram possíveis devido à habilidade dos cientistas em usar eficientemente computadores para gerar e processar grandes quantidades de dados.

Neste contexto, a elasticidade do modelo de computação na nuvem é particularmente

interessante para uma classe importante de aplicações de e-ciência que são caracterizadas por cargas de trabalho que requerem computação de alta vazão. Muitas destas aplicações podem ser paralelizadas trivialmente, através da quebra do trabalho a ser realizado em várias tarefas menores que podem ser processadas independentemente. Esta classe de aplicação é referenciada na literatura como aplicações "embaraçosamente paralelas" (*embarrassing parallel*) ou simplesmente "saco-de-tarefas" (BoT, do inglês *bag-of-tasks*) [Cirne et al. 2003]. Por exemplo, as simulações de Monte Carlo, que podem envolver a execução de milhares de cenários diferentes, podem ser paralelizadas simplesmente pela execução de cada cenário em uma unidade de processamento diferente. Aplicações que processam enormes quantidades de dados podem usualmente ser paralelizadas através da divisão dos dados entre um número de processos idênticos que executam a computação sobre cada bloco de dados independentemente; no final, pode ser necessário realizar algum tipo de consolidação dos processamentos individuais [Dean e Ghemawat 2008]. A renderização de imagens complexas e vídeos se encaixa bem nesta descrição. A lista de aplicações BoT é vasta e engloba não apenas usuários da academia, mas também da indústria e do governo. Além disso, a quantidade crescente de dados gerada e consumida pela sociedade moderna deve aumentar a pressão para executar eficientemente estas aplicações [Hey e Trefethen 2003].

Se o cliente que necessita executar uma aplicação BoT fosse capaz de requisitar de um provedor de computação na nuvem tantas máquinas virtuais quanto as necessárias para maximizar o nível de paralelização da execução da aplicação, isto lhe permitiria executar esta aplicação no menor tempo possível, sem que isso implicasse em um gasto extra com os recursos computacionais usados. A elasticidade do serviço oferecido por um provedor de IaaS é, obviamente, limitada pela quantidade física de recursos que ele dispõe. Acontece que, atualmente, esse limite é muito mais restritivo, uma vez que os provedores de computação na nuvem em operação restringem a quantidade de recursos que cada cliente pode demandar de cada vez a um número relativamente muito baixo, comparado com a capacidade dos provedores.

Usando simulação, nós fizemos uma análise para identificar as razões que levam os provedores de IaaS a impor limites que restringem a utilidade de seus serviços para a execução de aplicações que demandam elasticidade extrema. Os resultados das simulações, apresentadas no Capítulo 2, apontam que aumentos no limite imposto pelo provedor de IaaS levam

a impactos substanciais na sua lucratividade [Costa et al. 2011e; Costa et al. 2012e]. Um dos motivos é que quanto maior é o limite, maior é a capacidade da infraestrutura que os fornecedores precisam manter e, considerando uma taxa fixa de ociosidade, menor será a sua rentabilidade. Assim, os provedores públicos atuais de IaaS precisam limitar a quantidade de recursos que podem ser alocados concomitantemente por um mesmo usuário para que possam garantir uma disponibilidade de serviço suficientemente elevada para seus serviços e, ao mesmo tempo, manter os seus lucros em um nível aceitável.

Lidar com as demandas por elasticidade extremamente alta de aplicações HTC, BoT ou mesmo com *slashdot effects* ou *flash crowds* [Jung, Krishnamurthy e Rabinovich 2002], quando um grande número de usuários acessa simultaneamente um sítio Web que adquire uma popularidade instantânea, não é uma tarefa trivial. Proporcionar tal nível de flexibilidade traz desafios enormes para o planejamento de capacidade que precisa ser realizado pelos provedores de IaaS. Para dar suporte a este tipo de utilização, esses provedores provavelmente teriam que enfrentar níveis de ociosidade de suas estruturas maiores do que os que são observados hoje, com forte impacto em sua lucratividade. Dessa forma, é pouco provável que os provedores de IaaS atualmente em operação possam vir a oferecer um serviço mais adequado para os usuários que precisam executar aplicações que demandem uma elasticidade mais extrema. O resultado desta limitação é que existe uma faixa inteira de aplicações que ainda não está sendo bem atendida pelos serviços oferecidos atualmente pelos provedores de computação em nuvem.

Contando com modelos alternativos de provisionamento que permitam custos menores ou irrelevantes para a disponibilidade de recursos, os provedores de *JiT Clouds* podem proporcionar aos clientes com aplicações HTC, em geral, e BoT, em particular, os benefícios de uma maior amplitude na elasticidade da alocação de recursos: obter o menor tempo de processamento possível sem incorrer em aumento de custos.

1.2 Contribuições e Resultados

As principais contribuições deste trabalho são os seguintes:

- Investigação das causas que levam os provedores públicos de computação na nuvem a impor um limite estrito na quantidade de recursos que um único usuário pode adqui-

rir concomitantemente e análise de qual o impacto que eventuais aumentos no limite imposto apresentam sobre a lucratividade do provedor [Costa et al. 2012e];

- Uma proposta de uma nova arquitetura para computação distribuída que é ao mesmo tempo flexível e altamente escalável. Chamada de *OddCI - On-Demand Distributed Computing Infrastructure*, ela é suportada pela existência de um grande contingente de dispositivos que podem ser acessados simultaneamente através de uma rede de transmissão em *broadcast* [Costa et al. 2012d]. A técnica básica é, usando mensagens de controle enviadas pelo canal de *broadcast*, encontrar uma grande quantidade de processadores terceirizados disponíveis e configurá-los em conformidade e instantaneamente para o uso em infraestruturas computacionais dinâmicas voltadas para os requisitos de alta vazão de aplicações HTC;

- Implementação de um protótipo de sistema OddCI em um ambiente real de TV Digital para validação do conceito e obtenção de medições de campo [Costa et al. 2012c].

Os resultados de nossas experimentações mostram que é possível montar infraestruturas computacionais dinâmicas baseadas em recursos computacionais posicionados em praticamente todo o espectro de recursos terceirizados de baixa escala. Nos cenários mais desafiadores, envolvendo recursos de alta granularidade, alta volatilidade e alta dispersão, foi possível obter disponibilidade coletiva de dispositivos isolados para entregar vazão computacional com perdas máximas de 10% sob regimes de até 40% de volatilidade de nós, causada por falhas ou abandonos voluntários. Considerando o uso de recursos terceirizados não convencionais, como receptores de TV Digital de baixo custo, foi observada uma diferença relevante de capacidade computacional quando comparados com dispositivos convencionais, mesmo os de baixa granularidade. Entretanto, essa perda não se constitui em uma limitação técnica irreparável mas, tão somente, um aspecto mercadológico e circunstancial, passível de ser contornado com facilidade caso uma demanda para dispositivos mais potentes seja criada.

1.3 Organização

O restante deste documento está organizado em sete capítulos. No Capítulo 2 é feita uma contextualização do problema tratado nesta tese: a baixa amplitude da elasticidade oferecida pelos provedores atuais de computação na nuvem; no Capítulo 3 é apresentada uma breve fundamentação teórica para alguns dos aspectos envolvidos nesta pesquisa; no Capítulo 4 é apresentada uma abordagem alternativa para o provimento de infraestruturas para computação na nuvem baseada no uso de recursos terceirizados; no Capítulo 5 é feito o detalhamento de um mecanismo, chamado OddCI, para a montagem e operação de infraestruturas computacionais usando recursos de alta granularidade, alta dispersão e alta volatilidade; no Capítulo 6 é investigado o potencial de uso de recursos terceirizados não convencionais em sistemas OddCI, através da modelagem de uma implementação particular chamada OddCI-DTV, baseada em uma rede de receptores de TV Digital; no Capítulo 7 são apresentados alguns trabalhos relacionados com esta pesquisa; e, finalmente, encerramos o documento com o Capítulo 8, onde apresentamos um resumo dos resultados obtidos e uma discussão sobre direções para possíveis trabalhos futuros.

Capítulo 2

Baixa Amplitude da Elasticidade dos Provedores Atuais de Computação na Nuvem

Como discutido no capítulo anterior, os provedores públicos atuais de computação na nuvem precisam impor um limite estrito na quantidade de recursos que um único usuário pode adquirir concomitantemente. Neste capítulo nós fazemos uma análise que tenta identificar as razões que levam os provedores de IaaS a imporem limites que restringem a utilidade de seus serviços para a execução de aplicações BoT.

Nossa metodologia baseia-se no uso de simulação. Inicialmente, nós definimos um modelo simplificado de provedores de IaaS, apresentado na Seção 2.1, e um gerador de cargas de trabalho sintéticas apropriadas para o modelo proposto, discutido na Seção 2.2. Em seguida, nós apresentamos o modelo de simulação utilizado (Seção 2.3.1). Para instanciar o modelo de simulação de forma adequada, nós realizamos um projeto de experimento para identificar as variáveis aleatórias do modelo que têm um maior impacto na variável de resposta, e dessa forma definir os cenários de experimentação (Seção 2.3.2). Os resultados das simulações executadas que apresentamos na Seção 2.4 apontam que aumentos no limite imposto pelo provedor de IaaS levam a impactos substanciais na lucratividade do provedor. Dessa forma, é pouco provável que os provedores de IaaS atualmente em operação possam vir a oferecer um serviço adequado para os usuários que precisam executar aplicações BoT. Nas considerações finais deste capítulo (Seção 2.5), nós indicamos uma possível alternativa

para a implantação de um serviço de IaaS que possa atender apropriadamente essa classe de aplicações.

2.1 Um Modelo Simplificado de Provedor de IaaS

Assumindo que o serviço demandado por um cliente de um provedor de computação na nuvem ao longo do tempo é definido por uma sequência de tuplas $s_1, s_2, ...$, com $s_i = \langle \rho_i, \sigma_i, \delta_i \rangle$, onde ρ_i é a quantidade de recursos que foi solicitada na requisição de serviços s_i, σ_i é o momento em que o cliente deseja iniciar a usar os recursos e δ_i é a duração do intervalo de tempo para o qual os ρ_i recursos foram solicitados. A propriedade da elasticidade define que não há a imposição de nenhuma restrição para $\rho_i - \rho_{i-1}$ para qualquer $i, i > 1$, enquanto que a propriedade do pagamento pelo uso efetivo (do inglês *pay-as-you-go*) define que a fatura cobrada ao cliente por qualquer requisição s_i é uma função de $\rho_i \cdot \delta_i$.

A combinação das propriedades da elasticidade e do pagamento pelo uso efetivo, levam ao surgimento de uma terceira propriedade, chamada associatividade de custos [Fox 2011], a qual define que os clientes são tarifados com o mesmo valor para dois pedidos quaisquer s_i e s_j, tal que $\rho_i \cdot \delta_i = \rho_j \cdot \delta_j$.

Os provedores de computação na nuvem precisam, normalmente, fornecer garantias de qualidade de serviço (QoS, do inglês *Quality of Service*) que atendam plenamente os requisitos estabelecidos com os clientes que adquirem os seus serviços, expressos através de um acordo de nível de serviço (SLA, do inglês *Service Level Agreement*). Muitas dessas garantias são providas através da manutenção de capacidade excedente pelo provedor. Por outro lado, os custos do provedor são reduzidos pelas vantagens que a economia de escala pode proporcionar-lhe. Por exemplo, a concentração de sua estrutura em grandes centros de processamento de dados, dedicados e centralizados, e o compartilhamento de recursos físicos através da virtualização são estratégias cruciais para efetivamente oferecer serviços de uma forma economicamente viável. Sua competitividade também é baseada na capacidade de realizar uma multiplexação estatística de picos e vales no uso simultâneo de recursos por um grande número de clientes. Outra vantagem é o nível de automação atingido pelos provedores de computação na nuvem que, entre outras coisas, permite que eles reduzam substancialmente a relação de funcionários por servidores. Adicionalmente, os provedores

podem obter um aumento no nível de utilização dos seus serviços através da oferta de um portfólio de serviços que contemple diferentes modelos de precificação [Amazon 2011].

Dentre as muitas propriedade de QoS que um provedor de computação na nuvem precisa observar, neste trabalho nós iremos nos concentrar na disponibilidade de serviço (*service availability*), isto é, na probabilidade de que um cliente que solicita um serviço tenha o seu pedido plenamente atendido[1]. Esta propriedade não deve ser confundida com a disponibilidade de recurso (*resource availability*), que é representada pela probabilidade de que o serviço provido não irá falhar enquanto o cliente estiver usando-o. Em outras palavras, a disponibilidade de serviço é afetada quando um cliente solicita uma nova máquina virtual e o provedor é incapaz de instanciar o recurso demandado, enquanto que a disponibilidade de recurso é afetada sempre que uma máquina virtual que tenha sido instanciada para um cliente sofre uma falha. Observe que o SLA estabelecido entre o cliente e o provedor é normalmente focado na disponibilidade do recurso. Contudo, a disponibilidade do serviço é uma importante métrica para o provedor de IaaS, desde que um cliente cuja demanda é negada irá provalvelmente procurar outro provedor que atenda o seu pedido e pode nunca mais retornar para um provedor que apresenta uma disponibilidade de serviço limitada.

Seguindo o paradigma de computação na nuvem, um cliente de um provedor de IaaS solicita o provisionamento de recursos sempre que necessita deles. Se disponíveis, esses recursos são alocados para o cliente pelo provedor durante um certo período de tempo. Tipicamente, o cliente é quem define a duração de tal período, e devolve os recursos que lhe foram alocados quando os mesmos não forem mais necessários. Os provedores tarifam os clientes com base em um preço que está associado com um intervalo referencial minimo de alocação, de tamanho fixo (por exemplo, uma hora). Desta forma, os clientes são sempre cobrados pelo menor múltiplo de tal intervalo que é maior ou igual ao período de tempo pelo qual os recursos foram usados.

Nós estamos interessados em analisar o comportamento de um provedor de IaaS em um período de observação suficientemente longo de tamanho ΔT. Para simplificar o modelo, nós consideramos que este intervalo de tempo é discretizado em fatias menores de tempo de tamanho fixo (*time slots*), e que alocações e liberações de recursos são sempre realizadas no

[1] O foco em disponibilidade foi uma simplificação para tornar o modelo tratável, outras dimensões podem ser abordadas de maneira similar.

início das fatias de tempo. Nós modelamos um provedor de IaaS P como uma tupla:

$$P = \langle K, L, U, D, A, C_i, C_u, V, E \rangle \tag{2.1}$$

onde:

- K é a quantidade de recursos disponíveis no provedor, isto é, a sua capacidade;
- L é a quantidade máxima de recursos que pode ser alocada por um único cliente em cada fatia de tempo;
- U é o conjunto de usuários (clientes) registrados no provedor;
- D é a distribuição de demanda desses usuários;
- A é a estratégia de alocação de recursos usada pelo provedor;
- C_i é o custo incorrido pelo provedor para disponibilizar cada recurso individual por fatia de tempo, o qual é obtido pelo rateio da amortização do custo total de propriedade pelos recursos disponíveis e por todas as fatias de tempo que compreendem o período de amortização[2] [Li et al. 2009];
- C_u é o custo adicional incorrido pelo provedor sempre que um recurso é efetivamente usado em uma fatia de tempo, gasto somente quando cada recurso individual está sendo efetivamente usado. É baseado no conceito de custo de utilização proposto por Li *et al.* [Li et al. 2009] e considera que algum nível de proporcionalidade energética é praticado [Barroso e Hölzle 2007];
- V é o valor que é cobrado dos usuários pela utilização de um recurso por uma fatia de tempo ou fração;
- E é o encargo para o provedor por cada violação cometida na disponibilidade de serviço; ele pode ser tangível (ex. compensação contratual paga para o cliente) ou intangível (ex. dano na imagem do provedor). Neste trabalho nós consideramos apenas o aspecto tangível dos encargos por violações.

[2]Embora os custos descritos possuam um comportamento linear e representem uma simplificação dos custos reais, os quais apresentam um perfil mais complexo, esta simplificação fornece uma boa aproximação e atende às necessidades do nosso modelo.

Na próxima seção nós apresentaremos em detalhes como a demanda D dos usuários U de um provedor P é descrita. Por hora, vamos assumir que $d(u,t), 0 \leq d(u,t) \leq L, \forall u \in U, 1 \leq t \leq \Delta T$, é a quantidade de recursos demandada pelo usuário u em uma fatia de tempo t. Dependendo do padrão de demanda (D), da estratégia de alocação adotada (A), do limite de alocação por cliente (L) e da capacidade do provedor (K), cada usuário u que solicita $d(u,t)$ irá receber uma alocação de recursos associada que é expressa por $a(u,t), 0 \leq a(u,t) \leq d(u,t)$. Quando $a(u,t) < d(u,t)$ nos temos uma violação na disponibilidade de serviço do provedor. Assim, a quantidade total de violações em uma fatia de tempo t é dada por:

$$v(t) = \sum_{u \in U} 1 - \lfloor \frac{a(u,t)}{d(u,t)} \rfloor$$

Seja $\alpha(t)$ a capacidade alocada do provedor na fatia de tempo t. $\alpha(t) = \sum_{u \in U} a(u,t)$. Uma maneira de aferir a eficiência do provedor é medir o seu lucro no período de tempo considerado, representado em nosso modelo por:

$$\Lambda = \sum_{t=1}^{\Delta T} [(V - C_u) \cdot \alpha(t) - v(t) \cdot E] - K \cdot C_i \cdot \Delta T \tag{2.2}$$

2.2 Geração de Cargas de Trabalho Sintéticas para um Provedor de IaaS

Por causa da indisponibilidade de traços de execuções reais ou mesmo caracterizações da carga de trabalho de provedores de IaaS, foi necessário criar um gerador de cargas de trabalho sintéticas para definir a demanda imposta ao provedor em nossas simulações.

O uso total do sistema em cada fatia de tempo t, representado por $\alpha(t)$, é resultante do perfil de uso de cada usuário individual. Em princípio, todos os usuários podem, sob demanda e sem custos adicionais, se beneficiar da elasticidade inerente ao serviço e, em qualquer fatia de tempo, usar qualquer quantidade de recursos, de zero até o limite L imposto pelo provedor.

Considerando o comportamento do sistema no intervalo de tempo de duração ΔT, algumas categorias de usuários irão emergir. Uma classificação inicial dos usuários está relacio-

nada com o nível de demanda observada no período considerado: usuários ativos e usuários inativos. Os usuários ativos são aqueles que fizeram alguma demanda por recursos do sistema em um dado intervalo, ou seja, $d(u,t) > 0$ para algum valor de $t, 1 \leq t \leq \Delta T$. Os outros usuários são ditos inativos.

Seja U_a o conjunto de usuários ativos;

$$U_a = \{u | u \in U \wedge \exists t, 1 \leq t \leq \Delta T, d(u,t) > 0\}$$

O comportamento de cada categoria de usuário ativo é descrito através do uso das distribuições tradicionalmente associadas na literatura com classes de usuários e sessões de uso [Feitelson 2009; Talby 2006; Jain 1991]. Para a geração da carga de trabalho foi aplicada a abordagem de geração hierárquica, usando uma modelagem baseada no usuário [Feitelson 2009]. Esta técnica baseia-se na separação do comportamento dos usuários em três níveis: perfil da população/duração da sessão/atividade dentro da sessão, contemplando aspectos como localidade de amostragem (*locality of sampling*) [Feitelson 2009], além de auto-similaridade (*self-similarity*) [Feitelson 2009]. Com isto, é possível a inclusão na carga de trabalho gerada de longas permanências e ausências (cauda longa [Jain 1991]) e também de comportamentos regulares. O sistema modelado é do tipo fechado, com um número conhecido e finito de usuários ($|U_a|$).

A população de usuários ativos pode ser dividida em dois grupos, considerando a regularidade de demanda dos mesmos. Usuários ativos regulares são aqueles com uso ininterrupto. O conjunto de usuários regulares é descrito da seguinte forma:

$$U_r = \{u | u \in U_a \wedge \forall t, 1 \leq t \leq \Delta T, d(u,t) > 0\}$$

O conjunto de usuários eventuais (U_e) contém os usuários ativos não regulares:

$$U_e = U_a - U_r$$

Nós assumimos que os usuários regulares têm apenas uma sessão, cuja duração, em fatias de tempo, engloba pelo menos todo o intervalo ΔT considerado. Por outro lado, para os usuários eventuais o tempo de sessão é governado pelas seguintes variáveis aleatórias:

- $\tilde{o}$: duração (em fatias de tempo) de cada sessão de um usuário eventual, seguindo uma distribuição uniforme discreta com limite inferior l_o e limite superior u_o [Jain 1991]; e

- $\tilde{i}$: intervalo entre sessões, seguindo uma distribuição Pareto discretizada com parâmetros k_i e s_i [Jain 1991].

Dentro de cada sessão, o usuário pode estar "em atividade" ou em "espera" (*think time*), que indicam, respectivamente, se o usuário está efetivamente usando recursos, ou não. O comportamento de cada usuário em atividade pode ser definido pela quantidade de recursos que ele utiliza, pela duração deste uso e também pelo tempo que ele fica sem usar os recursos do sistema. Desta forma, cada atividade pode ser caracterizada pela tupla:

$$\mathcal{A} = \langle r, n, e \rangle$$

onde r e n representam a quantidade de recursos requisitados por fatia de tempo e a duração da atividade em número de fatias de tempo, respectivamente, e e representa o tempo de espera até a próxima fatia de tempo na qual o usuário estará em atividade. A mudança na quantidade de recursos, embora possível, implica no início de outra atividade.

A seguir, serão descritos os perfis de uso de cada categoria de usuário da nossa população.

O perfil de uso dos usuários regulares foi modelado de uma forma simplificada. Usuários regulares apresentam atividades ininterruptas (sem espera) que duram uma fatia de tempo. Em cada sessão o número de recursos demandados é baseado na variável aleatória $\tilde{m}$ com distribuição normal, média τ e variância σ, onde τ é o *ticket* médio dos usuários regulares, dado por:

$$\tau = \frac{\sum_{t=1}^{\Delta t} \sum_{u \in U_r} a(u,t)}{\Delta T \cdot |U_r|}$$

O perfil de atividade dos usuários regulares é definido como:

$$\mathcal{A}_{regular} = \langle \tilde{m} \sim N(\tau, \sigma), 1, 0 \rangle$$

Esta abordagem modela possíveis aumentos ou diminuições em solicitações individuais dos usuários regulares. Entretanto, a multiplexação estatística da demanda regular conduz a variações pouco significativas na utilização total dos usuários regulares em cada fatia de tempo. Mudanças mais abruptas no comportamento de usuários regulares que afetam este relacionamento serão tratadas adiante.

O comportamento "em atividade" dos usuários eventuais, por sua vez, é baseado em três variáveis aleatórias:

- $\tilde{s}$: quantidade de recursos alocados em cada atividade, seguindo uma distribuição uniforme discreta entre 1 e L [Jain 1991];
- $\tilde{d}$: duração (em fatias de tempo) de cada atividade, seguindo uma distribuição exponencial discreta com média λ_d [Jain 1991]; e
- $\tilde{t}$: intervalo (em fatias de tempo) entre atividades (*think time*), seguindo uma distribuição exponencial discreta com média λ_t [Jain 1991].

O perfil de atividades dos usuários eventuais é definido como:

$$\mathcal{A}_{eventual} = \langle \tilde{s} \sim U(1, L), \tilde{d} \sim E(\lambda_d), \tilde{t} \sim E(\lambda_t) \rangle$$

Dois perfis particulares de usuários eventuais foram também modelados para cobrir as seguintes situações: a) usuários regulares apresentando uma demanda não usual por recursos motivada por *flashcrowds* ou *flashmobs* em seus serviços, com intensidade variável [Jung, Krishnamurthy e Rabinovich 2002]; e, b) usuários eventuais com utilização intensiva e sensível ao tempo (ex.: usuários de aplicações BoT) [Sevior, Fifield e Katayama 2010] que sempre consomem todos os recursos disponíveis. Estes perfis são definidos da seguinte forma:

$$\mathcal{A}_{flashmob} = \langle U(\tau + 1, L), \tilde{d} \sim E(\lambda_d), \tilde{t} \sim E(\lambda_t) \rangle$$

$$\mathcal{A}_{BoT} = \langle L, \tilde{d} \sim E(\lambda_d), \tilde{t} \sim E(\lambda_t) \rangle.$$

A inclusão do perfil *flashmob* teve como principal objetivo permitir a representação, no modelo proposto, da ocorrência esporádica de grandes e repentinos aumentos no tráfego para um determinado *website* que possui, normalmente, uma demanda conhecida e controlada. Em geral, são incidentes isolados e raros mas de grande impacto para os serviços atingidos.

2.3 Descrição dos Experimentos

O principal objetivo dos experimentos de simulação é observar: i) a capacidade mínima necessária para atendimento de todas as solicitações para um determinado nível de disponibi-

lidade de serviço; ii) a ociosidade do sistema em cada cenário; e, iii) o resultado operacional do provedor com diferentes limites.

Em seguida apresentaremos como o modelo de simulação foi implementado e como os cenários de simulação foram instanciados.

2.3.1 Implementação do Modelo de Simulação

Para ser resolvido por simulação, o modelo proposto foi implementado usando a ferramenta Möbius [Deavours et al. 2002]. Esta plataforma permite a realização de simulação de eventos discretos e resolução numérica ou analítica de modelos de sistemas que podem ser descritos em uma variedade de formalismos.

Um dos formalismos suportados permite a composição de modelos em uma estrutura de árvore, na qual cada folha da árvore pode ser um modelo atômico, descrito em um dos outros formalismos suportados, ou outro modelo composto. Cada nó da árvore que não é uma folha é classificado ou como um nó *Join* ou como um nó *Replicate*. Um nó do tipo *Join* é usado para compor dois ou mais submodelos através do compartilhamento de estado, enquanto um nó do tipo *Replicate* é usado para construir um modelo consistindo de um determinado número de cópias idênticas do seu submodelo filho.

Para representar os usuários ativos de um provedor IaaS, nós usamos este formalismo para a criação do modelo composto *ActiveUsers* (Figura 2.1). Este modelo contém quatro submodelos atômicos, modelados usando o formalismo *Stochastic Activity Network* (SAN), representando os quatro perfis de usuários descritos: *Regular*, *Eventual*, *FlashMob* e *BoT*. O uso dos nós *Replicate* permite a criação do número desejado de instâncias de cada perfil de usuário definido e também o compartilhamento de estado entre as instâncias de um mesmo tipo de submodelo. O nó *Join*, por sua vez, permite o compartilhamento de estado entre instâncias de submodelos de tipos diferentes. Desta forma, a carga de trabalho sintética foi construída através da atividade autônoma e combinada de uma instância do submodelo *Regular*, cuja demanda em cada fatia é multiplicada por $|U_r|$, e um total de $|U_e|$ instâncias dos submodelos *Eventual*, *FlashMob* e *BoT*, criadas de acordo com a distribuição de atividade configurada para cada tipo de perfil.

Por exemplo, o submodelo Eventual, mostrado na Figura 2.2, representa o comportamento de um usuário do perfil *Eventual*. Conforme descrito na seção anterior, um usuário

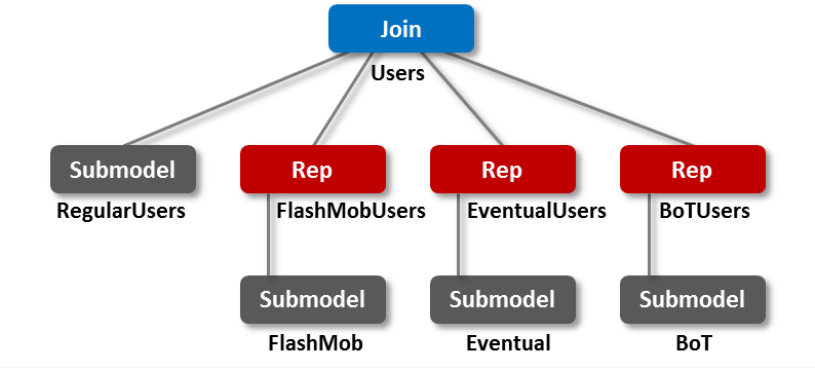


Figura 2.1: O Modelo Composto dos Usuários Ativos de um Provedor IaaS

consome recursos da nuvem através de uma série de estágios. Estes estágios foram modelados em um submodelo SAN como lugares (*places*) e lugares extendidos (*extended places*), representados na figura por círculos azuis e laranja, respectivamente. Cada lugar mantém um contador (representado por *tokens*) que expressam o estado corrente do usuário naquele estágio. Os portões de entrada (*input gates*), representados por triângulos vermelhos, são usados para inspecionar estes estados e habilitar (ou não) a transição do sistema através da execução de atividades temporizadas (barras verticais). Cada atividade temporizada tem uma duração que impacta na dinâmica do sistema modelado e também uma distribuição (e parâmetros associados) que regula o seu comportamento. Os portões de saída (*output gates*), representados pelos triângulos pretos, são executados após o tempo de duração de uma atividade temporizada ter sido completada e permite a alteração do estado do sistema através da alteração do número de *tokens* nos lugares. Os arcos (linhas pretas) sinalizam o fluxo de transição de estágios. Cada usuário de perfil *Eventual* é inicializado randomicamente em um dos estágios possíveis (*OnSession* ou *OffSession*), os quais são controlados pelo lugar *On*. Após a inicialização, as atividades *OffTime* e *OnTime* começam a regular a alternância do usuário em sessões de uso e períodos de inatividade, controlados pelas variáveis aleatórias $\tilde{o}$ e $\tilde{i}$, respectivamente. Uma nova atividade para o usuário em sessão é atribuída (conforme descrito no perfil *Eventual* e usando as variáveis aleatórias $\tilde{d}$ e $\tilde{s}$) através da porta de saída *SetActivity* após um período de espera (*think time*) ser cumprido. A duração esperada de cada período de espera é gerida pela atividade temporizada *NewThinkTime* (variável aleatória $\tilde{t}$). O lugar *ActivityControl*, por sua vez, controla a duração de cada atividade individual, fatia a fatia de tempo, através da atividade temporizada *NewCycle*.

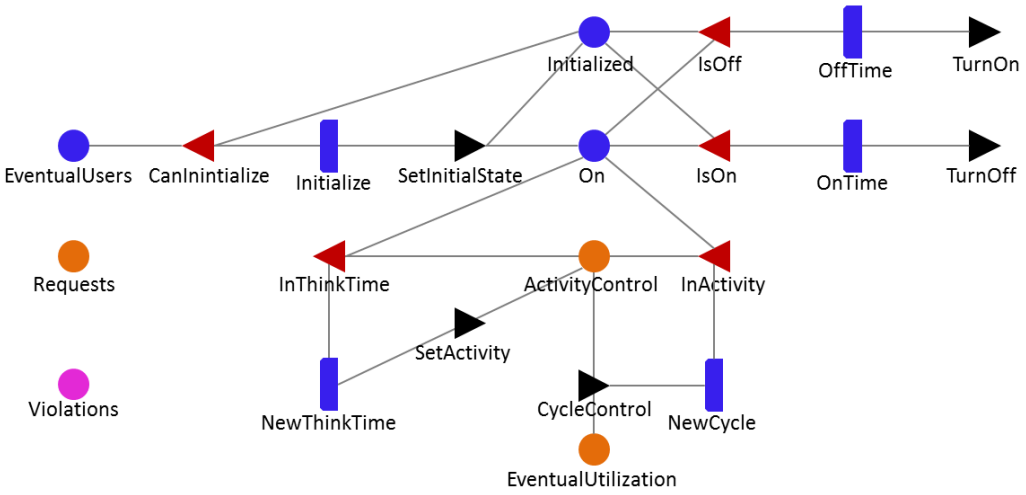


Figura 2.2: O modelo atômico (SAN) de um usuário do perfil *Eventual*

Os outros submodelos — *Regular* (Figura 2.3), *FlashMob* (Figura 2.4) e *BoT* (Figura 2.5) — possuem modelagem similar [3].

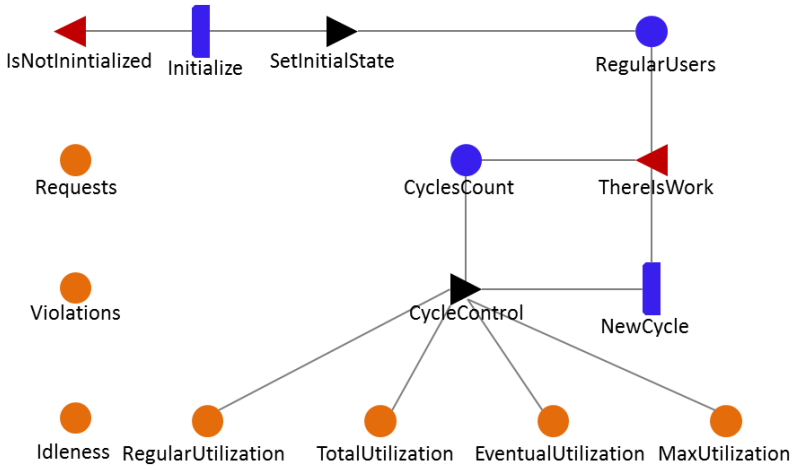


Figura 2.3: O modelo atômico (SAN) de um usuário do perfil Regular

A dinâmica da população de usuários configurada é quem dirige a alocação de recursos do provedor de IaaS. Nós assumimos uma algoritmo de alocação *First-Come-First-Service* muito simples, que sempre atribui a quantidade de recursos que são demandados por cada solicitação do usuário enquanto houver capacidade livre suficiente disponível. As variáveis de resposta produzidas pelo modelo de simulação foram a capacidade alocada em cada fatia

[3] O modelo Möbius completo usado nos experimentos de simulação realizados para esta análise pode ser encontrado no sítio http://www.lsd.ufcg.edu.br/~rostand/IaaSModel.zip.

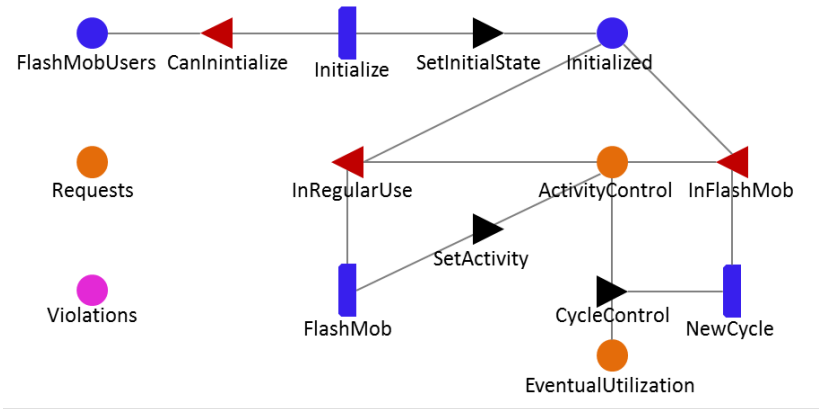


Figura 2.4: O modelo atômico (SAN) de um usuário do perfil *FlashMob*

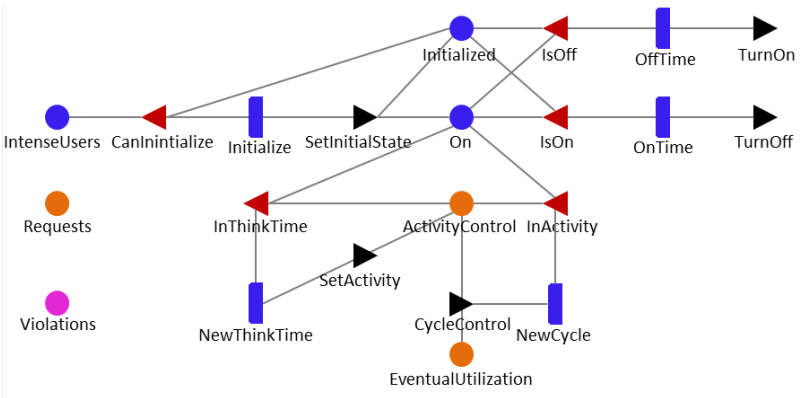


Figura 2.5: O modelo atômico (SAN) de um usuário do perfil *BoT* (Intenso)

Fator	**Baixo**	**Alto**	**Efeito Estimado**	**Soma dos Quadrados**	**% Cont.**
A: Limite superior u_o (em fatias) para $\tilde{o}$	36	108	$0,06$	$0,03$	$6,53$
B: Limite inferior k_i (em fatias) para $\tilde{i}$	120	360	$-0,03$	$0,01$	$1,66$
C: Média λ_d (em fatias) para $\tilde{d}$	$0,0625$	$0,1875$	$0,07$	$0,04$	$8,83$
D: Média λ_t (em fatias) para $\tilde{t}$	$0,125$	$0,375$	$-0,02$	$0,00$	$0,81$
E: L (em quantidade de recursos)	20	100	$0,21$	$0,37$	$77,05$

Tabela 2.1: Fatores, níveis e efeitos para DoE 2^k fatorial ($k = 5$)

de tempo ($\alpha(t)$) e o número de violações por fatia de tempo ($v(t)$).

Os experimentos de simulação são executados usando o simulador Möbius simplesmente fornecendo as configurações adequadas para os diversos parâmetros do sistema, incluindo aqueles exigidos pela modelagem da carga de trabalho que acaba de ser apresentada.

2.3.2 Parâmetros do Sistema

Para atribuição dos parâmetros do sistema foram usadas duas estratégias: projeto de experimento (DoE, do inglês *Design of Experiment*) e varredura de parâmetros. A parte dos parâmetros relacionada com a geração da carga sintética e associada com as distribuições descritas na Seção 2.2 foi tratada através de um DoE do tipo 2^k fatorial [Jain 1991]. Através do DoE foi possível analisar o efeito dos parâmetros das variáveis aleatórias $\tilde{o}$ (duração da sessão), $\tilde{i}$ (intervalo entre sessões), $\tilde{s}$ (duração da atividade), $\tilde{t}$ (*think time*) e também do valor de L sobre uma das variáveis de resposta do sistema: a utilização máxima do sistema em um dado intervalo $(max(\alpha(t)) \ \forall \ t, 1 \leq t \leq \Delta T)$. Os níveis atribuídos para o DoE são apresentados na Tabela 2.1.

Foram conduzidas várias repetições dos 32 experimentos para obter médias com intervalo de confiança de 95%. A contribuição de cada fator está exibida na Tabela 2.1, com destaque para o fator predominante, L, o qual teve contribuição de $77,05\%$. A única interação relevante (acima de $0,5\%$) foi BC que apresentou uma contribuição de $2,53\%$. Como resultado da análise dos efeitos através de ANOVA [Jain 1991], o F-Value de $158,6521$ implica que o modelo é significativo. O R^2 ajustado indica que o modelo explica $96,83\%$ da variação observada e o R^2 de predição está dentro de $0,20$ do R^2 ajustado, representando uma boa

Parâmetro	**Valor**
Duração da Sessão ($\tilde{o}$)	$l_o = 1$ hora e $u_o = 72$ horas
Intervalo entre Sessões ($\tilde{i}$)	$k_i = 240$ horas e $s_i = 2$
Duração da Atividade ($\tilde{d}$)	$\lambda_d = 0.125$ (8 horas)
Espera entre Atividades ou *think time* ($\tilde{t}$)	$\lambda_t = 0.25$ (4 horas)
ΔT	8.760 horas (1 ano)
Número de Usuários Ativos ($\|U_a\|$)	$\{ 625; 1.250; 2.500; 5.000 \}$
Percentual de Atividade Eventual	$\{ 25\%; 35\%; 45\%; 55\%; 65\%; 75\%; 85\%; 95\% \}$
Percentual de Usuários com Perfil *FlashMob*	1%
Percentual de Usuários com Perfil *BoT*	$\{ 10\%; 15\%; 20\%; 25\% \}$
Limite (L)	$\{ 20; 30; 40; 50; 60; 70; 80; 90; 100 \}$
Ticket Médio (τ)	2 recursos

Tabela 2.2: Parâmetros Usados na Simulação

capacidade de predição do modelo[4].

De acordo com os resultados, a variação dos quatro primeiros fatores não afetou o comportamento da variável de resposta que ocorreu em função da variação de L.

Para a realização das simulações, os valores dos quatro parâmetros com impacto muito baixo foram ajustados para a média entre os respectivos níveis "Alto" e "Baixo" usados no DoE. Para os parâmetros *Percentual de Atividade Eventual*, *Percentual de Usuários com Perfil BoT*, *Número de Usuários Ativos* e L foi aplicada uma estratégia de varredura de parâmetros. Foi adotado um *ticket* médio de 2 recursos, que representa apenas 10% do limite para alocação de automática de recursos praticado pelo principal provedor de IaaS em operação. Além disso, foi considerada uma participação discreta, de apenas 1%, dos usuários com Perfil *FlashMob* na população simulada. A Tabela 2.2 mostra como o sistema foi configurado para os experimentos.

2.3.3 Validação e Verificação

Considerando uma perspectiva operacional e concreta, Miser *et al.* [Miser 1993] define o termo "validação" como "o processo pelo qual cientistas asseguram a si mesmos e aos outros

[4]Maiores detalhes sobre este estudo, incluindo os gráficos de diagnóstico, cubo e interação, podem ser encontrados no sítio http://www.lsd.ufcg.edu.br/∼rostand/IaaSModel.zip.

que uma teoria ou modelo é uma descrição de um fenômeno determinado, sendo adequado ao uso para o qual será aplicado". Em outras palavras, a validação do modelo conceitual permite determinar se as teorias e suposições nas quais o modelo se baseia são corretas e se a representação que o modelo faz do problema é adequada para os propósitos do modelo [Sargent 1998].

Landry *et al.* [Landry, Malouin e Oral 1983] já haviam contribuído de maneira significativa para o entendimento desta questão, argumentando que a validação não é uma fase separada e independente do processo de construção do modelo, mas é interligada e contínua ao longo de todo o ciclo de desenvolvimento, propondo atrelar as atividades de validação ao processo de construção do modelo, estabelecendo o conceito de "processo de modelagem e validação".

Considerando que a melhor maneira de provar que o modelo proposto de provedor de IaaS é eficaz é colocando-o em prática, o ideal seria se pudéssemos dispor de dados ou estatísticas de nuvens reais para apoiar as nossas suposições. No entanto, não tivemos conhecimento, durante a realização dessa pesquisa, de qualquer conjunto público de dados que possuíssem informações suficientes para dar suporte a uma validação do nosso modelo conceitual. Possivelmente, estudos semelhantes podem ter sido feitos pelos provedores de nuvens para sua própria análise de lucratividade e planejamento de capacidade, mas os mesmos não têm demonstrado interesse em tornar esses dados disponíveis publicamente. Só recentemente, a Google divulgou alguns de seus rastros (traces), mas eles apresentam informações bastante limitadas e estão muito fragmentados, não sendo aplicáveis no nosso caso.

Assim, uma das suposições mais relevantes que usamos, a de que o padrão de utilização dos usuários individuais pode ter reflexos mais amplos na infraestrutura do provedor, foi baseada no uso de uma carga de trabalho sintética. Como é sabido hoje que o comportamentos dos usuários não tendem a seguir, necessariamente, uma certa distribuição, esta assumpção poderia fazer o modelo ter, em certa medida e dependendo da sua parametrização, algum tipo de viés ou conduzir a resultados previsíveis.

Com o intuito de aferir a robustez do modelo, nós realizamos uma análise de sensibilidade para verificar o impacto de nossas suposições de distribuição sobre os resultados produzidos pelo modelo. Neste sentido nós executamos todos os experimentos de simulação aplicando ao modelo de geração hierárquica baseado no usuário que foi utilizado dois con-

juntos distintos de distribuição, ambos referenciados na literatura. No primeiro deles, usamos as distribuições pareto e exponencial, como descritos por Feitelson [Feitelson 2009] e Jain [Jain 1991] e no segundo, nós acrescentamos ainda mais imprevisibilidade ao modelo, considerando um esquema de distribuição hiper-exponencial de dois estágios, como sugerido por Coffman e Wood para modelar o comportamento de usuários interativos em sistemas mais antigos [Coffman Jr. e Wood 1966].

Os resultados observados, para ambos os casos, são essencialmente os mesmos e, o mais importante, nos conduziu para as mesmas conclusões. Isto é, provavelmente, devido à dinamicidade complexa do modelo baseado no usuário utilizado, no qual a carga de trabalho é constituído por uma combinação do comportamento individual de cada usuário simulado.

A implementação do modelo conceitual foi realizada usando abstrações de alto nível através do formalismos de redes de atividades estocásticas usando uma ferramenta de modelagem e simulação validada e madura, o Möbius [Deavours et al. 2002]. Isto facilitou a realização da verificação da corretude da implementação, que foi feita através da revisão criteriosa dos modelos atômicos e compostos criados e testes de aceitação, e da validação operacional, realizada com variação de parâmetros e análise dos traços correspondentes do Möbius para aferir a acurácia das saídas produzidas.

2.4 Resultados e Análise

No primeiro experimento, o objetivo foi observar como a lucratividade do provedor era impactada com o aumento do limite imposto pelo provedor (L). Nesse experimento nós consideramos uma situação em que a disponibilidade de serviço do provedor deve ser mantida em 100%. Para este fim , a capacidade (K) simulada foi configurada de forma que, para qualquer fatia de tempo t, sempre é possível alocar recursos para um usuário u que tenha uma demanda positiva ($d(u,t) > 0$) e, portanto,

$$a(u,t) = d(u,t), \forall u \in U \wedge 1 \leq t \leq \Delta t$$

.

Dessa forma, considerando a Equação 2.2, como as penalidades serão nulas e a receita líquida da execução de uma mesma carga de trabalho é constante, o lucro do provedor é

afetado apenas pela capacidade que precisa ser mantida para atender o nível de disponibilidade desejado. Para garantir condições similares de carga do sistema, o número de usuários ativos foi mantido constante para este experimento em 5.000 usuários. Entretanto, foi feita uma varredura dos parâmetros *Percentual de Atividade Eventual* e *Percentual de Usuários com Perfil BoT* para simular diferentes cenários de atividade regular e eventual e diferentes participações dos usuários com perfil *BoT*. Esta classe de usuários é especialmente interessante para esta análise porque possuem cargas de trabalho de alto volume e sensíveis ao tempo e tendem a consumir todo o limite máximo de alocação de recursos permitido (L).

Para cobrir todas as combinações dos parâmetros de entrada foram realizadas 288 simulações. Cada cenário foi repetido até que os níveis de confiança esperados fossem atingidos (95% de intervalo de confiança). A resposta de interesse foi a capacidade máxima alocada ($max(\alpha(t))$) observada em todas as fatias de tempo de cada configuração do sistema simulado, já que esta define a capacidade mínima necessária para garantir 100% de disponibilidade de serviço durante o período de simulação. Parte dos resultados obtidos estão exibidos graficamente na Figura 2.6.

Como pode ser observado, mesmo assumindo uma população de tamanho constante, a capacidade mínima necessária aumenta à medida que o limite é incrementado. Esta demanda por maior capacidade já está presente mesmo em cenários onde a atividade regular é dominante com 25% de usuários eventuais, dos quais somente 10% possuem o perfil *BoT* (Figura 2.6(a)). Onde a atividade eventual é mais preponderante, com 95% de todos os usuários, o aumento necessário da capacidade instalada chega a ser de mais de três vezes, à medida em que o limite aumenta de 20 para 100. Considerando um cenário com 25% de usuários com perfil *BoT* (Figura 2.6(b)), a capacidade mínima necessária atinge o triplo com 75% de atividade eventual, atingindo picos de aumento de quatro vezes quando tal atividade atinge 95% e o valor do limite é configurado para 100.

É interessante notar que quando o limite é configurado para 20 no cenário com 10% de usuários com perfil *BoT*, o aumento do percentual de usuários eventuais conduz a um decréscimo na capacidade necessária, o que está em oposição ao que acontece quando são impostos grandes valores para o limite (área azul claro na Figura 2.6(a)). Uma inspeção mais detalhada sobre os resultados da simulação revelou que isto acontece porque, neste caso particular, a distribuição da demanda de 10% de usuários *BoT* acaba sendo diluída na grande

massa de usuários eventuais. Quando o percentual de usuários com perfil *BoT* aumenta, este fenômeno não é mais relevante e a pressão causada por este tipo de usuário começa a ser sentida na capacidade necessária mesmo para valores baixos do limite (Figura 2.6(b)).

A Figura 2.7 mostra uma perspectiva diferente, na qual o percentual de usuários com perfil *BoT* varia de 10% a 25% em dois cenários de percentagem de utilização eventual (25% e 75%). Novamente, é possível observar um aumento consistente na capacidade mínima necessária em ambos os cenários, influenciada tanto pelo aumento do valor do limite quanto pelo aumento no número de usuários *BoT*. É possível ver que a percentagem de usuários eventuais tem um impacto mais forte na capacidade mínima necessária quando combinada com o percentual de usuários com perfil *BoT* e com o aumento no limite de recursos que pode ser alocado simultaneamente por um cliente.

Uma segunda análise permitiu observar como o incremento na capacidade instalada afeta o nível de utilização do sistema. Usando os valores de $max(\alpha(t))$ obtidos no experimento anterior como a capacidade instalada do provedor (K), nós obtivemos a ociosidade apresentada pelo sistema. A ociosidade é representada pela razão entre a quantidade total de recursos usada durante o período ΔT e a capacidade total disponível no mesmo período:

$$\frac{\sum_{t=1}^{\Delta T} \alpha(t)}{K \cdot \Delta T}$$

A Figura 2.8 ilustra a ociosidade encontrada em dois cenários: 10% e 25% de usuários com perfil *BoT*.

Os resultados indicam uma variação da ociosidade proporcional à variação do limite e da percentagem de usuários eventuais, apresentando entre 20% e 65% de capacidade ociosa em todas as combinações simuladas de atividade eventual e perfil *BoT*.

A Figura 2.9 mostra, para um cenário com 10% de usuários *BoT* e diferentes níveis de atividade eventual, a evolução do aumento percentual da capacidade mínima necessária para evitar violações, e a correspondente ociosidade observada, à medida em que o valor do limite foi sendo aumentado nos experimentos realizados. Como pode ser visto na Figura 2.9(a), a capacidade mínima necessária mantém um expansão quase constante, em termos percentuais, em resposta ao incremento na percentagem de usuários eventuais e no valor do limite imposto. Por outro lado, como pode ser visto na Figura 2.9(b), o percentual de ociosidade aumenta seguindo uma padrão diferente de evolução: quanto maior é a percentagem de

usuários eventuais, menor é o aumento percentual do nível de ociosidade atingido quando o valor do limite aumenta. No caso de 95% de usuários eventuais, o aumento percentual da ociosidade observado fica abaixo de 1% em cada patamar de limite, o que conduz a um aumento total abaixo de 5% quando o limite varia de 20 até 100. O mesmo comportamento também foi observado em cenários com outras percentagens de usuários *BoT*.

Isto acontece porque quando o número de usuários eventuais é grande, a ociosidade já é alta, mesmo para pequenos valores do limite, como pode ser visto na Figura 2.8. Por outro lado, este comportamento mostra que, embora o aumento no limite conduza a impactos consideráveis sobre os níveis de ociosidade, o aumento do número de usuários eventuais tem impacto ainda maior sobre a ociosidade do sistema.

Este aumento proporcional da ociosidade com o aumento do limite tem reflexos significativos nos custos do provedor. A necessidade de aumentar a capacidade mínima necessária tem impacto nos investimentos iniciais para o provedor (CAPEX), enquanto que o correspondente aumento nos níveis de ociosidade tem impacto nos seus custos operacionais (OPEX). Considerando o preço cobrado pelo provedor de IaaS que é o atual líder do mercado [Amazon 2010] e usando a expressão para cálculo do lucro (Equação 2.2), foi realizada uma terceira análise. Foram aplicadas diferentes margens de lucro aos valores obtidos nos experimentos anteriores para identificar o ponto a partir do qual a operação do provedor se torna equilibrada, ou seja, sem lucro nem prejuízo, em cada configuração. Foi observado que à medida que o limite é incrementado o ponto de equilíbrio da operação só é alcançado quando a margem de lucro também é aumentada, com reflexos diretos na competitividade do provedor. Na Figura 2.10, pode ser visto que a margem de lucro necessária para igualar receitas e despesas varia de 40% até quase 60% no maior valor considerado para o limite, para uma variação de 25% até 75% de atividade eventual e com apenas 10% de usuários com perfil *BoT*.

Nos experimentos anteriores, foi fixado o tamanho da população em 5.000 usuários (o número máximo de instâncias do modelo que a ferramenta utilizada suportou simular). A fim de avaliar o impacto que o tamanho da população poderia ter nos resultados, os mesmos experimentos foram repetidos para quantidades diferentes de usuários ativos. Mantidas as mesmas condições de limite e perfis de atividade, as curvas observadas são bastante similares para todas as quantidades simuladas de usuários ativos (Figura 2.11). Esta é uma indicação de que a economia de escala pode não desempenhar um papel direto de melhoria

na rentabilidade dos provedores de IaaS quando um mesmo valor de L é utilizado.

Os resultados apresentados até agora consideram um cenário em que violações não ocorrem. Embora a disponibilidade de serviço deva ser sempre muito alta, raramente é rentável mantê-la 100%. Dado este fato, também realizamos experimentos para avaliar como um nível de disponibilidade de serviço mais relaxado iria impactar na ociosidade do sistema e, como resultado, no seu custo operacional. Nesses experimentos, nós gradualmente reduzimos a capacidade mínima necessária para que nenhuma violação ocorresse, identificada nos experimentos anteriores, e, para cada redução realizada, medimos as violações introduzidas.

A disponibilidade de serviço para vários valores de limite, em uma população com apenas 35% de usuários eventuais, é mostrada na Figura 2.12(a). Pode-se observar que a redução de capacidade tem efeitos mais dramáticos sobre a disponibilidade do serviço para os valores mais baixos de limite. Isso é explicado pelo fato de que essas são as configurações que apresentem menor ociosidade, e, portanto, tem menos flexibilidade para reduções da capacidade instalada. As capacidades ociosas calculados para as mesmas situações são mostradas na Figura 2.12(b), onde o efeito já discutido pode ser melhor visualizado.

Note que estas simulações permitem a um provedor de serviços realizar uma análise invertida para identificar o valor mais adequado para o limite L de forma a atingir um nível desejado de margem de lucro. Para isso, o provedor deve escolher o valor de L que melhor equilibre a sua capacidade ociosa resultante (custos de disponibilidade) e o nível de disponibilidade do serviço (custos de violações).

Nossos experimentos mostram que, enquanto a demanda de usuários regulares é permanente e previsível, o seu crescimento é benéfico para a rentabilidade do provedor, uma vez que não impõe um risco de superdimensionamento da infraestrutura. Assim, o lucro do provedor pode ser afetado negativamente pela demanda que vem de usuários eventuais, a qual pode resultar em aumento da inatividade da infraestrutura, se não for controlada. Isso é agravado quando os usuários eventuais são grandes consumidores de recursos e fazem demandas pontuais muito grandes. Observou-se que os usuários com utilização eventual e intensa forçam a capacidade mínima necessária e aumentam a inatividade do sistema, aumentando os custos operacionais do provedor. Desta forma, não só a atribuição de um limite para a alocação de recursos é necessária, mas também o valor atribuído pode ter um impacto significativo sobre os investimentos em infraestrutura para garantir um nível adequado de

disponibilidade de serviço para o provedor.

2.5 Considerações Finais

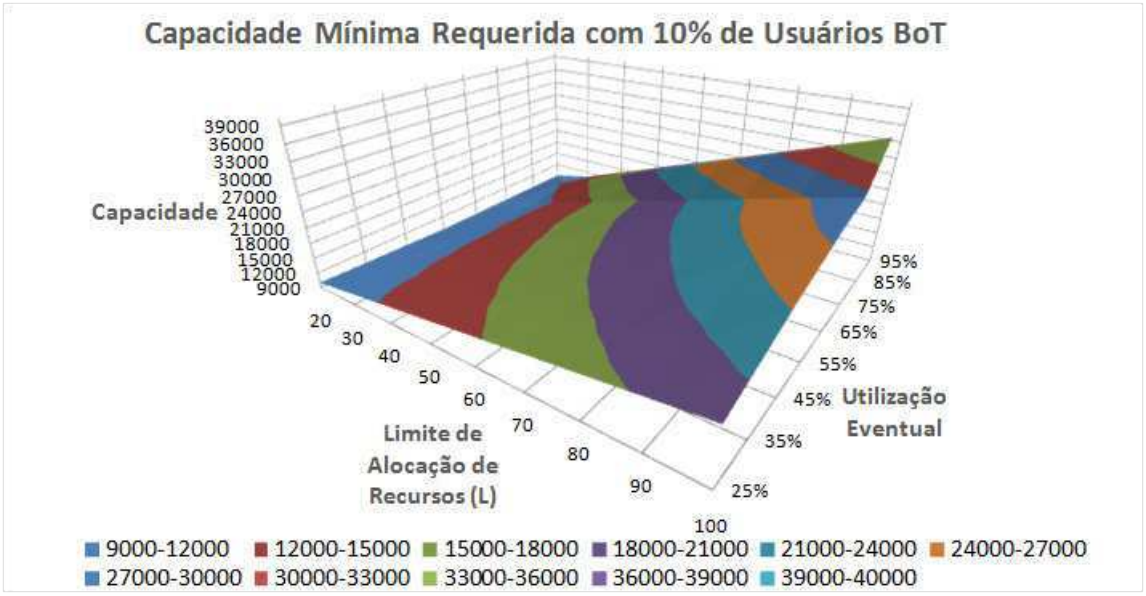
Neste capítulo foram analisadas as razões que levam os fornecedores atuais IaaS a impor limites muito restritivos sobre a quantidade de recursos que um cliente pode adquirir simultaneamente. Nossa avaliação utiliza um modelo de simulação para um provedor de IaaS, que é alimentado com uma carga de trabalho sintética, o que permitiu a simulação de uma ampla variedade de cenários. O uso de modelo mais próximo da realidade nos pareceu a opção mais adequada para este estudo. Para mitigar a complexidade do modelo e a inexistência de dados de campo, usamos técnicas como o *design* de experimento, para identificar as variáveis independentes mais importantes, e a varredura de parâmetros, para a instanciação de um amplo espectro de cenários. Obtivemos resultados consistentes em todos os cenários simulados.

A análise dos resultados aponta que é necessária a atribuição de um limite para a quantidade de recursos que pode ser simultaneamente alocada por um usuário, a fim de manter a disponibilidade do serviço suficientemente elevada e a um custo razoável para o provedor. O valor real para esse limite vai variar de provedor para provedor dependendo de sua própria avaliação de onde situa-se o equilíbrio, mas os nossos resultados indicam que ele tende a não ser muito maior do que os valores atualmente praticados que se enquadram no intervalo de algumas dezenas. Observou-se também que os usuários com perfis *Eventual* e *BoT* pressionam a capacidade mínima necessária e aumentam a ociosidade do sistema, aumentando os custos operacionais do provedor. Além disso, mantidos o mesmo perfil da população e o mesmo valor de limite, a dinâmica do sistema independe da quantidade de usuários e não constitui, portanto, um contexto onde a economia de escala possa significar uma melhoria direta.

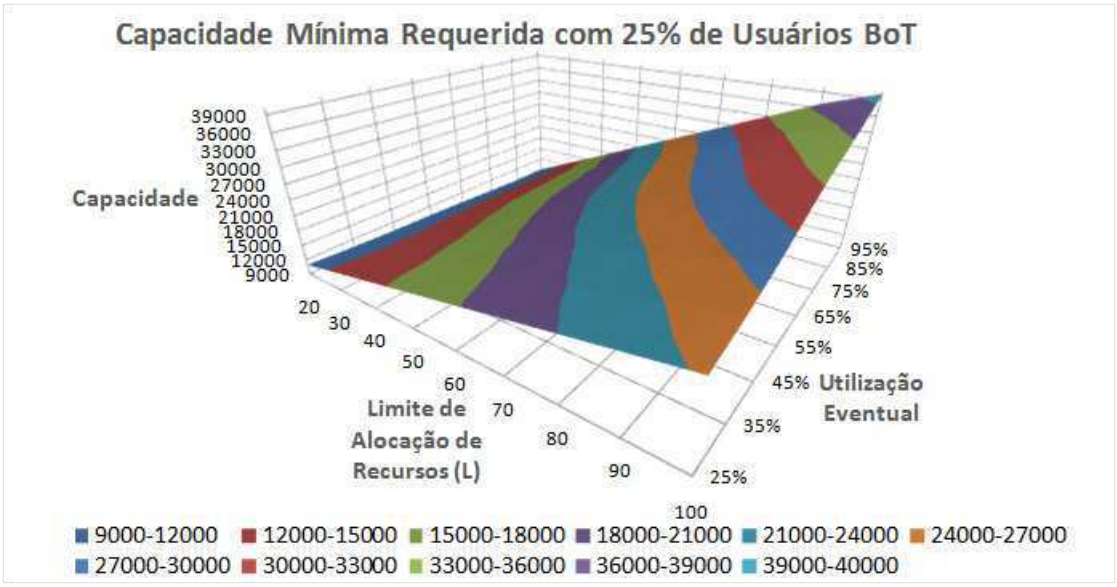
Os resultados ajudam a entender a necessidade do uso de um limite e como o seu impacto na lucratividade do provedor está diretamente relacionado com o padrão de utilização da população de usuários, nos fazendo concluir que algumas categorias de usuários/aplicações que se beneficiariam de uma elasticidade mais ampla, continuarão sendo mal servidas se o modelo atual de provisionamento de recursos for mantido.

Neste sentido, os próximos capítulos deste trabalho serão dedicados à investigação de

formas alternativas para minimizar os custos envolvidos com o aumento da capacidade dos provedores públicos de computação na nuvem para lidar apropriadamente com a demanda de usuários eventuais ávidos por recursos, tais como aqueles que precisam executar grandes aplicações científicas *BoT*. Estes custos são um dos principais obstáculos para a oferta de elasticidade em condições mais flexíveis, mesmo que ainda limitada, mas que permitam que classes de aplicações de uso intenso possam se beneficiar das vantagens do modelo de computação na nuvem. A descoberta, federação e revenda de recursos terceirizados pode representar um caminho promissor, pois se baseia no aproveitamento, sob demanda, de capacidade ociosa existente em contextos onde os custos de instalação e disponibilidade n so recaem sobre o operador da federação.

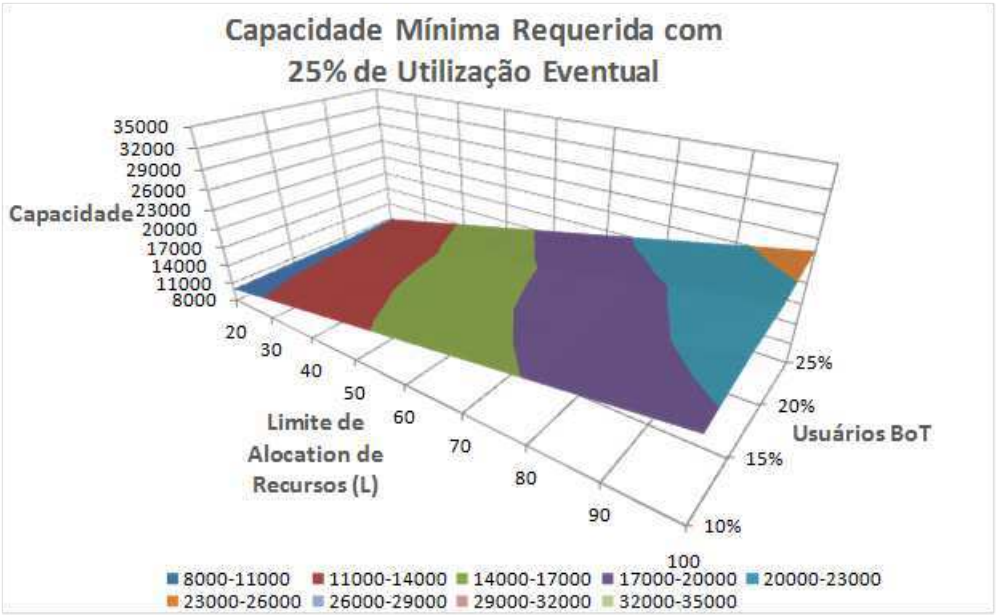


(a)

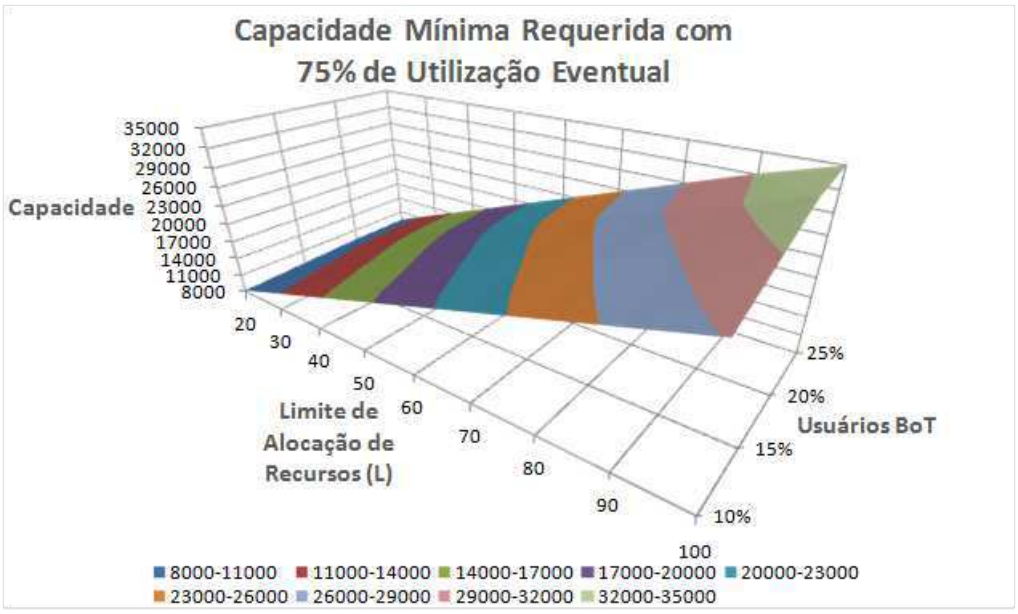


(b)

Figura 2.6: Capacidade mínima necessária para atingir 100% de disponibilidade quando variando o limite (L) e a atividade eventual para dois cenários de usuários com perfil *BoT* (10% and 25%)

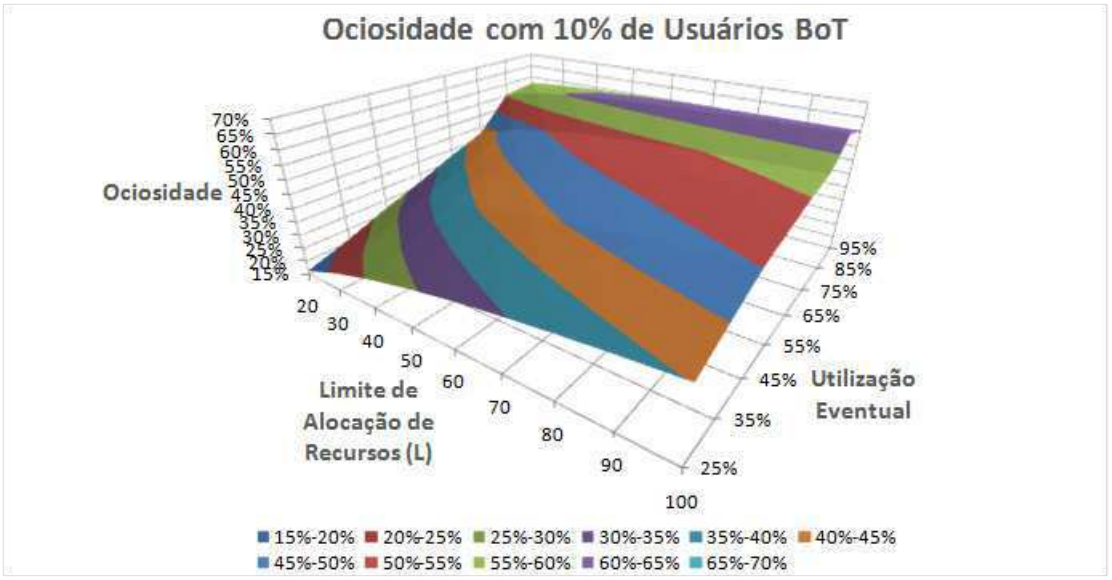


(a)

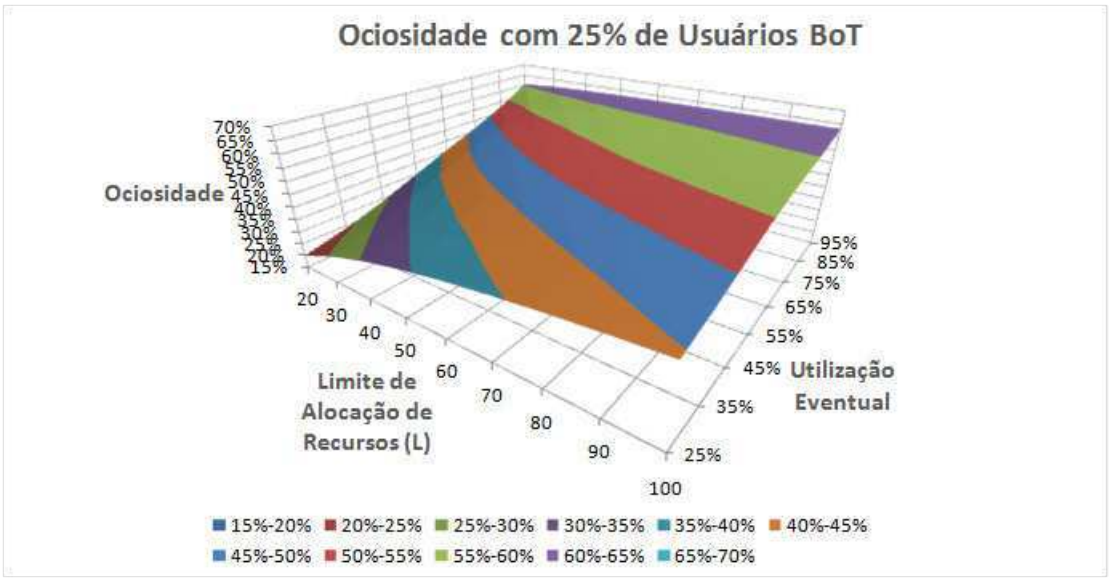


(b)

Figura 2.7: Capacidade mínima necessária para 100% de disponibilidade quando variando o limite (L) e a percentagem de usuários com perfil *BoT* para diferentes cenários de utilização eventual

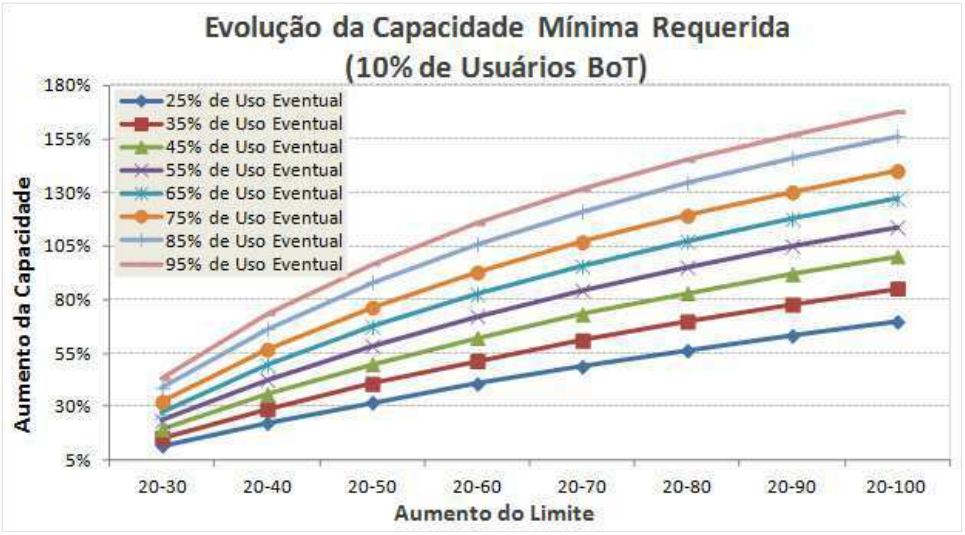


(a)

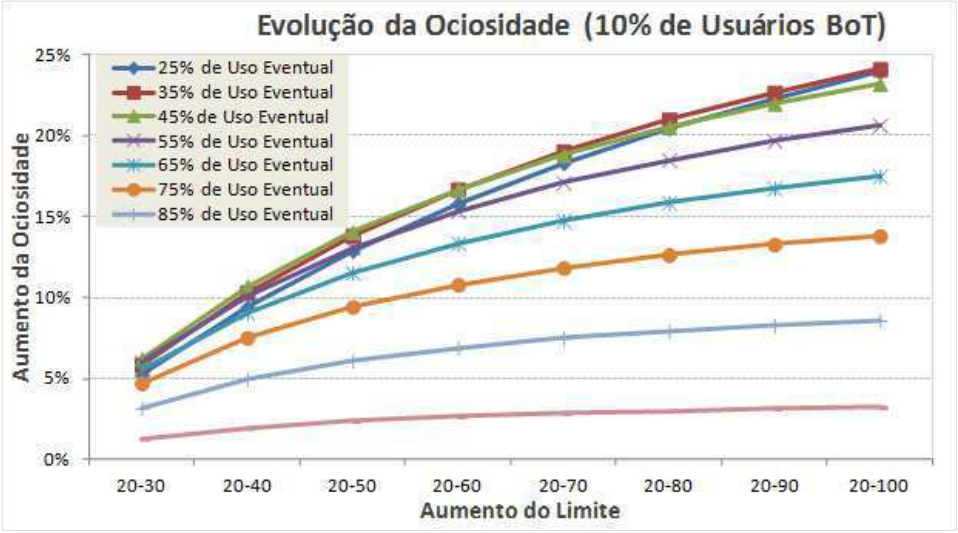


(b)

Figura 2.8: Ociosidade observada quando variando o limite (L) e a percentagem de usuários eventuais para diferentes cenários de usuários com perfil BoT



(a)



(b)

Figura 2.9: Evolução da capacidade mínima necessária e da ociosidade observada quando variando o limite (L) e a percentagem de usuários eventuais para um cenário de 10% de usuários com perfil *BoT*

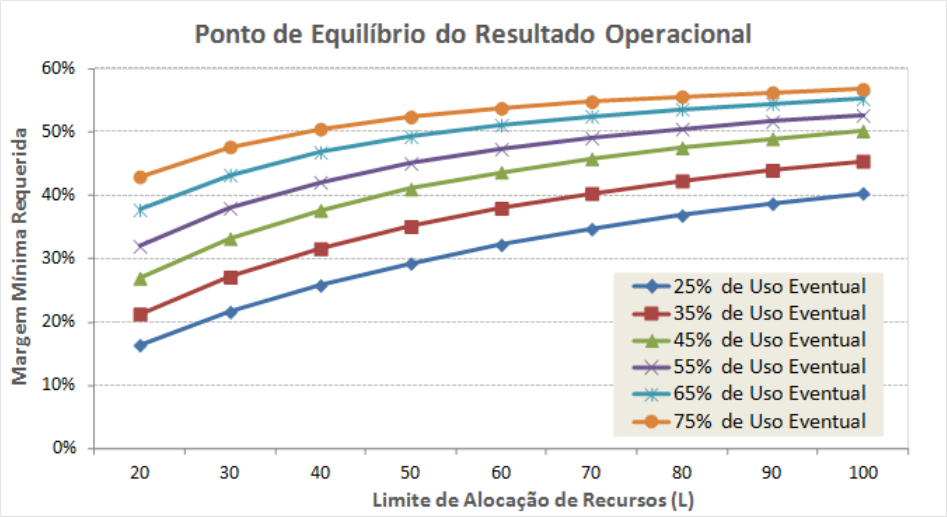


Figura 2.10: Equilíbrio do resultado operacional quando variando o limite (L) e a percentagem de usuários eventuais para um cenário de 10% de usuários com perfil *BoT*

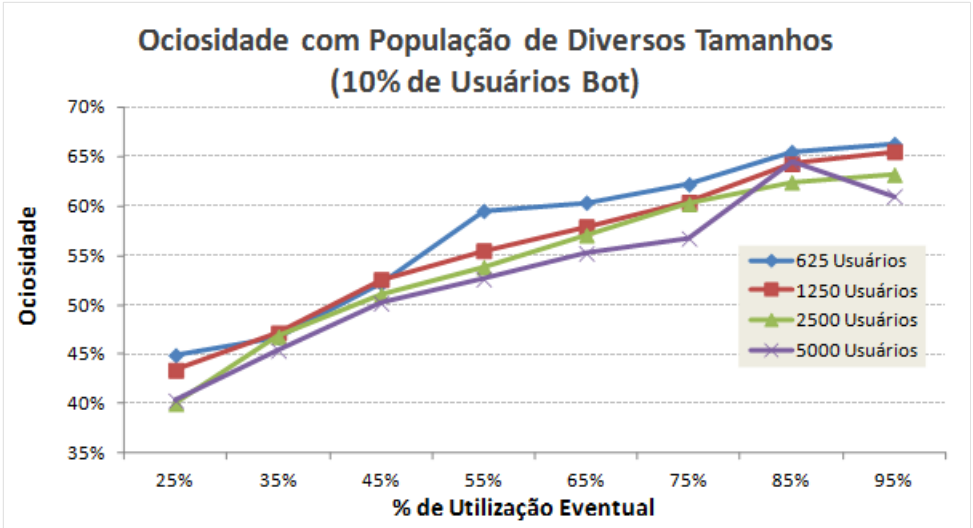
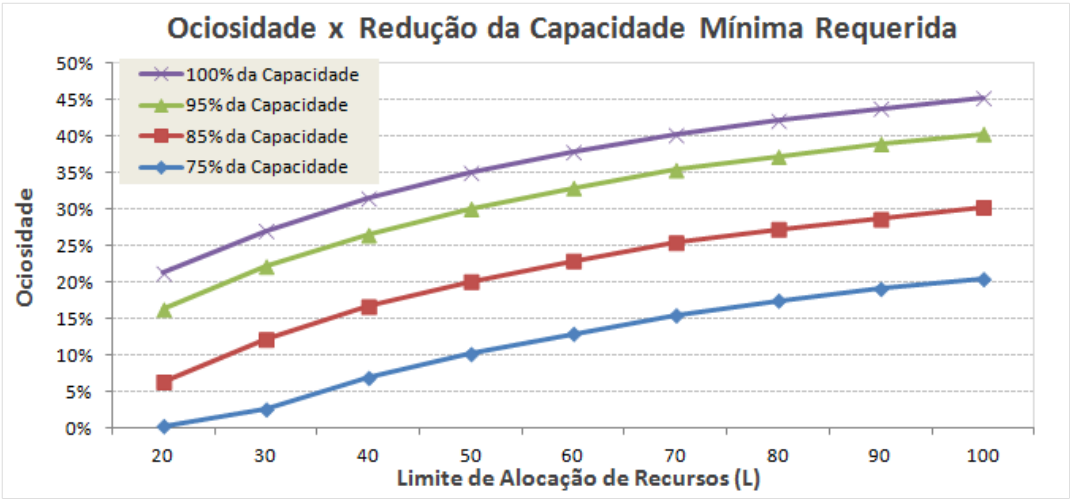
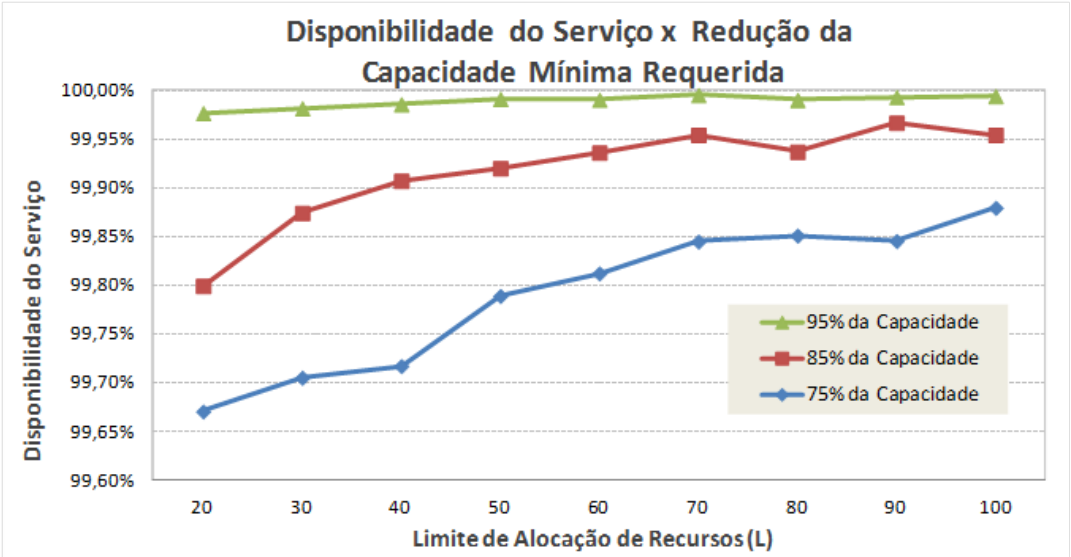


Figura 2.11: Ociosidade para populações de diferentes tamanhos



(a)



(b)

Figura 2.12: Nível de disponibilidade de serviço e ociosidade após uma redução na capacidade mínima necessária para atingir 100% de disponibilidade de serviço

Capítulo 3

Fundamentação Teórica

3.1 Computação na Nuvem

Computação na nuvem (do inglês *cloud computing*) é um modelo de oferta e gestão de serviços de Tecnologia da Informação (TI) que traz grandes modificações na forma como todos os atores envolvidos no negócio de TI passam a atuar. Virtualização é a tecnologia de base que permitiu o surgimento da computação na nuvem. Essa tecnologia permite que as infraestruturas de TI possam ser consolidadas e melhor aproveitadas, reduzindo custos em todas as dimensões, desde custos de aquisição de *hardware* e *software*, passando por custos com instalações físicas e energia elétrica, e principalmente os custos com pessoal especializado para dar suporte à operação da infraestrutura de TI. Quanto maior é a infraesturtura de TI de uma organização, maiores serão as possibilidades de economia com a utilização de virtualização. A economia de escala associada à tecnologia de virtualização, permitiu que a consolidação dos serviços de TI ultrapassasse as fronteiras de uma organização, e pudessem ser vendidas como um serviço para outras organizações, menos capacitadas tecnologicamente, ou com infraestruturas de TI menores [Amazon 2010].

Entre as várias definições de computação na nuvem, uma que começa a ganhar relevância é aquela proposta pelo Instituto Nacional de Padrões e Tecnologia do Departamento de Comércio do Governo dos Estados Unidos da América (NIST). Segundo o NIST [Hogan et al. 2011], "computação na nuvem é um modelo que habilita o acesso ubíquo, conveniente, sob demanda, através de uma rede de computadores, a um conjunto de recursos compartilhados (ex. redes, servidores, dispositivos de armazenamento, aplicações e serviços) que

podem ser rapidamente provisionados e liberados com um esforço mínimo de gerência ou de interação com seus respectivos provedores."

A partir dessa definição é possível listar algumas características fundamentais presentes em sistemas de computação na nuvem:

- **Acesso remoto**: os serviços de computação na nuvem são disponibilizados na Internet e são acessados utilizando mecanismos padronizados para diferentes tipos de plataforma cliente, como PDAs, *smart phones* e computadores pessoais.

- **Auto-serviço sob demanda**: o consumidor de um serviço de computação na nuvem é capaz de provisionar o serviço oferecido de forma automática e quase instantânea, no momento que ele julgar conveniente. Isso significa que o consumidor é capaz de demandar, configurar, utilizar, e desmobilizar os serviços oferecidos pelo provedor de computação na nuvem sem a intervenção de um humano.

- **Serviços mensuráveis**: os serviços ofertados por um provedor de nuvem computacional são passíveis de medição acurada. A forma desta medição depende do tipo de serviço; assim, a quantidade de serviço de processamento oferecido pode ser medida por hora de utilização, a de armazenamento em disco por *bytes* armazenados, enquanto que a utilização de um serviço de *e-mail* pode ser medida por número de mensagens recebidas ou enviadas, apenas para citar alguns exemplos. Essa característica permite ao usuário requisitar e utilizar apenas a quantidade de serviço necessária para atender suas necessidades.

- **Elasticidade**: uma das características mais importantes de um provedor de computação na nuvem é sua capacidade de escalar os recursos provisionados de acordo com as necessidades e a qualquer tempo. Em momentos de pico de demanda o sistema deve poder prover mais recursos, passado o pico os recursos provisionados podem ser liberados, diminuindo o custo para o consumidor. A impressão para o consumidor deve ser que os recursos são infinitos e estão sempre a sua disposição.

- **Aglomeração de recursos**: um provedor de computação na nuvem oferece serviços sobre um aglomerado de recursos computacionais que através de sistemas de gerência

> de virtualização são dinâmicamente atribuídos e compartilhados para atender a demanda de serviços dos consumidores. Tipicamente essa demanda é heterogênea, permitindo que os recursos liberados por um consumidor em um momento sejam atribuídos para outros consumidores que necessitam de mais recursos naquele momento.

Computação na nuvem pode ser implantada seguindo diferentes modelos, dependendo de onde a infraestrutura física é mantida e da relação entre provedores e consumidores de serviço. Esses modelos de implantação são discutidos em detalhes na Seção 3.1.1. Por sua vez, independentemente do modelo de implantação, o paradigma de computação na nuvem é adequado para prover uma grande variedade de serviços, desde aqueles já tradicionalmente ofertados no modelo cliente-servidor até novos serviços de infraestrutura computacional como rede, armazenamento e processamento, levando ao conceito de "tudo-como-um-serviço" (EaaS, do inglês *everything-as-a-service*). Considerando essa nomeclatura, os três principais modelos de serviço de computação na nuvem são: infraestrutura (IaaS, do inglês *Infrastructure-as-a-Service*), plataforma (PaaS, do inglês *Platform-as-a-Service*) e *software* (SaaS, do inglês *Software-as-a-Service*). Esses modelos de serviço são discutidos em detalhes na Seção 3.1.2.

3.1.1 Modelos de Implantação

Um sistema de computação na nuvem tem pelo menos dois tipos de atores: consumidores e provedores. Em linhas gerais, consumidores são aqueles que se beneficiam das características de rápida provisão e liberação de recursos, elasticidade e pagamento por tempo ou quantidade de recursos efetivamente usados. Os provedores por outro lado, precisam se preocupar com a adequada implantação e operação dos mecanismos que permitem que eles ofereçam serviços para seus consumidores com essas características de uma forma sustentável.

Um dos requisitos fundamentais para permitir a operação sustentável do provedor de computação na nuvem é a habilidade de atender uma grande quantidade de consumidores, utilizando a tecnologia de virtualização para isolar aplicações e consolidar servidores, e a economia de escala para reduzir seus custos de operação. Dependendo da relação entre os

consumidores e a organização que mantém o sistema de computação na nuvem, existem quatro modelos de implantação possíveis. O sistema de computação na nuvem é dito *privado* quando os consumidores do serviço são todos vinculados à mesma organização que provê o serviço. Quando o serviço é oferecido apenas para consumidores vinculados a um conjunto bem definido de organizações, trabalhando de forma consorciada, o sistema é dito *comunitário*. Quando os consumidores não têm qualquer vínculo com a organização que provê o serviço, a menos de uma relação consumidor/provedor de serviço, o sistema é dito *público*. Finalmente, quando o sistema é uma nova combinação formada pela associação de infraestruturas de tipos diferentes, ele é dito *híbrido*.

Cada modelo de implantação tem suas características particulares, vantagens e desvantagens. Entretanto, algumas características são comuns a todos os modelos [Badger et al. 2011]. Em primeiro lugar, todo sistema de computação na nuvem depende do correto funcionamento e da segurança provida pela rede de computadores que permite o acesso dos consumidores ao serviço. Além disso, os consumidores tipicamente têm pouco ou nenhum controle sobre a localização física e a distribuição de cargas de trabalho dos servidores que executam o serviço. Por conta disso, as aplicações dos consumidores estão sujeitas aos riscos associados com a execução de múltiplas aplicações sobre o mesmo servidor físico [Oberheide, Cooke e Jahanian 2008]. Por sua vez, estes riscos estão relacionados com falhas no *software* utilizado pelo provedor para implementar virtualização e com erros de configuração das políticas de segurança definidas pelos provedores.

As características listadas acima ressaltam duas questões importantes relacionadas com o *controle* e a *visibilidade* que o consumidor tem sobre a infraestrutura que provê o serviço na nuvem. Por controle entende-se a habilidade de decidir, com alta confiabilidade, quem pode ter acesso a que dados e programas do consumidor. Por visibilidade entende-se a habilidade de monitorar, com alta confiabilidade, o estado dos dados e programas do consumidor, e como estes estão sendo acessados por terceiros. Dependendo do modelo de implantação adotado, controle e visibilidade precisam ser relaxados em maior ou menor grau. Os riscos e as proteções legais associadas com esse relaxamento precisam ser bem compreendidos pelos consumidores dos serviços oferecidos na nuvem.

Em infraestruturas convencionais, controle e visibilidade são definidos através da criação de barreiras de acesso, sobre as quais políticas de segurança podem ser configuradas e asse-

guradas. Duas barreiras de acesso bastante conhecidas são as redes virtuais privadas (VPNs, do inglês *virtual private networks*) e os *firewalls*. Estes criam perímetros de segurança, dividindo os consumidores em duas classes, quais sejam: aqueles que estão dentro do perímetro e que têm acesso irrestrito a todos os recursos (ex. dados, programas, etc.) protegidos pela barreira de acesso, e aqueles que estão fora do perímetro e que portanto estão sujeitos às restrições de acesso implementadas pela barreira.

3.1.2 Modelos de Serviço

Infraestrutura como um Serviço (IaaS)

O serviço de IaaS é baseado na oferta de recursos virtualizados de processamento, armazenamento e rede. Esses recursos são abstraídos através de máquinas virtuais (VMs, do inglês *virtual machines*), que podem ser administradas através de comandos enviados através da rede para o provedor utilizando um *shell* remoto seguro (SSH, do inglês *secure shell*) ou interfaces remotas gráficas utilizando os protocolos RDP (*Remote Desktop Protocol*) ou RFB (*Remote Framebuffer Protocol*). Em geral o assinante está livre para escolher o sistema operacional desejado oferecendo uma imagem de VM completa ou escolhendo entre aquelas pré-definidas pelo provedor. Os serviços de IaaS podem atender assinantes que desejam hospedar suas aplicações na nuvem ou servir de base para a oferta de serviços de mais alto nível, como PaaS e SaaS, tanto em nuvens privadas como em nuvens públicas.

Podemos olhar para IaaS como uma evolução do serviço tradicional de hospedagem ou locação de máquinas em centro de dados (*data centers*). A diferença fundamental é que IaaS permite que a alocação de recursos computacionais seja feita de forma simplificada, dinâmica e, sobretudo, elástica, enquanto que, no modelo tradicional de hospedagem e locação, o conjunto de recursos alocados é mais estático e as mudanças nos termos de serviços contratados demandam um processo mais demorado, envolvendo negociação entre humanos. Em IaaS o assinante tem o maior nível de controle sobre o serviço, entretanto ele fica responsável por operar, atualizar e configurar os recursos com objetivo de atingir os níveis de desempenho, de segurança e de confiabilidade desejados. O provedor deve manter um gerenciador de nuvem (a partir do qual os assinantes gerenciam seus recursos); um gerenciador de *cluster* (que recebe os pedidos de alocação do gerenciador de nuvem); e

gerenciadores para os equipamentos propriamente ditos, que na maioria dos casos é um supervisor (*hypervisor*) que permite iniciar, terminar e reinicar máquinas virtuais. O provedor ainda deve oferecer armazenamento persistente de dados e conectividade estável.

Os candidatos naturais para utilizar IaaS são instituições que buscam uma alternativa a manter seus próprios centros de dados e a evitar investimentos antecipados em infraestrutura. A adoção do modelo de IaaS nem sempre leva a uma redução no custo total incorrido pelo assinante, entretanto a flexibilidade para adaptar os custos operacionais à demanda é um grande atrativo. Outro atrativo é a possibilidade de hospedar aplicações legadas na nuvem, já que em muitos casos é possível customizar o ambiente de execução, tipicamente expresso pela adequada configuração da imagem de uma VM. Entretanto ao se optar por um modelo de serviço de IaaS alguns pontos devem ser considerados: dependência de uma conexão de rede segura e confiável, o que nem sempre pode ser garantido; exposição das vulnerabilidades do sistema legado e do sistema operacional executando nas VMs; segurança no processo de autenticação; e quais são as garantias de isolamento tanto da solução de virtualização quanto da rede usadas pelo provedor.

Atualmente existe um grande número de provedores de IaaS. Ainda que muito semelhantes entre si em relação aos modelos de cobrança adotados, os serviços ofertados e alguns outros pontos podem apresentar pequenas diferenças.

Plataforma como um Serviço (PaaS)

Um provedor de PaaS oferece um ambiente que permite ao assinante criar e desenvolver aplicações elásticas capazes de atender um grande número de requisições de maneira facilitada e sem ter que se preocupar com os detalhes da plataforma de execução [Rimal, Choi e Lumb 2009; Foster et al. 2008]. Comparado com o desenvolvimento de aplicações convencionais, essa abordagem ajuda a diminuir o tempo de desenvolvimento, ao oferecer ferramentas e serviços, além de possibilitar a rápida escalabilidade sob-demanda das aplicações desenvolvidas.

Um assinante de PaaS recebe basicamente duas classes de serviço. Uma das classes de serviço compreende um ambiente de desenvolvimento e de gerência de aplicação que atende as equipes de desenvolvimento, testes e implantação. Esta é a interface para o serviço de PaaS propriamente dito. Uma segunda classe de serviços atende os clientes do assinante

do serviço de PaaS que utilizarão as aplicações desenvolvidas e hospedadas no provedor de PaaS. A idéia é que o assinante do serviço de PaaS submeta uma aplicação, e então o provedor desse serviço se encarrega de alocar recursos, instalar, configurar e então disponibilizar o acesso à aplicação de seu assinante através da rede. Após a aplicação estar em funcionamento, o provedor do serviço de PaaS também oferece aos seus assinantes ferramentas para administrar e monitorar as aplicações por eles instaladas, possibilitando o acesso a informações sumariadas sobre a aplicação, como por exemplo quantidade de acessos, carga de CPU, uso de memória, instâncias da aplicação na infraestrutura, etc.

As ferramentas de desenvolvimento e as aplicações desenvolvidas são acessadas através de um navegador Web, o que implica em uma necessidade reduzida de instalação de *software* tanto para o assinante quanto para seus clientes. Essa característica facilita questões de gerência de *software*, entretanto é necessário atenção aos riscos de segurança decorrentes de tal interface. Outra vantagem oferecida pelo modelo de PaaS é que ainda que os dados estejam fisicamente espalhados pela rede do provedor, do ponto de vista do assinante, toda gerência de dados, incluindo os de desenvolvimento, é realizada de forma centralizada.

Um risco existente em PaaS é a falta de padronização entre os provedores. Em geral, a aplicação desenvolvida na plataforma de desenvolvimento de um determinado provedor não poderá operar em outro. Da mesma maneira, o formato dos dados armazenados por essa aplicação pode ter que ser totalmente reestruturado para se adaptar a outro provedor.

Software como um Serviço (SaaS)

Um provedor de SaaS oferece uma ou mais aplicações que podem ser acessadas pelos assinantes, ou usuários finais, através de um portal Web. Todas as atividades de manutenção da infraestrutura de execução e gerência, bem como desenvolvimento e atualização das aplicações são de responsabilidade do provedor. Assim, em geral o assinante não tem controle sobre a infraestrutura de execução e tem acesso a um número limitado de configurações da aplicação.

Uma característica importante de SaaS é que não há necessidade de instalação e manutenção de nenhum *software* no lado do cliente a não ser um navegador. Também quase não existe necessidade de processamento local já que todos os dados são mantidos na infraestrutura de computação na nuvem, onde são processados. Uma das grandes vantagens deste

modelo de serviço é a possibilidade de acesso universal, inclusive através de dispositivos móveis. Hoje existe uma enorme quantidade de aplicações bastante populares disponibilizadas através de um modelo de SaaS, como por exemplo: serviços de correio eletrônico como o Gmail e o Yahoo; redes sociais como Facebook, Twitter e Orkut; carga e descarga de fotos e vídeos com Flickr ou Youtube; ferramentas de produtividades como o Microsoft Office Web e GoogleDocs; e também no campo de gestão de empresas com aplicativos de gestão de relacionamento com os clientes (CRM, do inglês *Customer Relationship Management*) oferecido pela Salesforce.

3.2 Escalabilidade e Elasticidade para Computação de Alta Vazão

Computação paralela é uma tecnologia chave para permitir o processamento tempestivo da quantidade crescente de dados que está sendo gerada por sensores, experimentos científicos, modelos de simulação e, ultimamente, como um efeito da era de digitalização que a nossa sociedade como um todo está experimentando. De fato, algumas das cargas de trabalho (*workloads*) que precisam ser processadas são tão grandes, que a única maneira viável para lidar com elas, em um tempo razoável, é quebrar o processamento em uma determinada quantidade de tarefas menores, e executá-las em paralelo no maior número disponível de processadores. Em uma classificação bastante ampla, notadamente quando se consideram as diferenças entre as características das cargas de trabalho, a computação paralela é normalmente dividida em Computação de Alta Performance (HPC, do inglês *High Performance Computing*) e Computação de Alta Vazão (HTC) [Litzkow, Livny e Mutka 1988].

Obviamente, paralelismo em larga escala só pode ser alcançado se houver unidades de processamento disponíveis e um nível relativamente elevado de independência entre as tarefas que compõem a aplicação paralela. Felizmente, muitas das cargas de trabalho das aplicações paralelas podem ser mapeadas em tarefas que podem ser processadas de forma completamente independente uma das outras, compondo uma classe de aplicações conhecida como "*bag-of-tasks*" (BoT) [Cirne et al. 2003]. O fato de que as tarefas de uma aplicação BoT são totalmente independentes, não só faz o agendamento trivial, mas também faz com que a tolerância a falhas seja muito mais fácil, já que um mecanismo de repetição simples

pode ser usado para recuperar tarefas que eventualmente falhem durante a execução. Como consequência, as aplicações BoT são menos exigentes com a qualidade do serviço suportado pela infraestrutura computacional subjacente.

A vazão obtida quando se executam aplicações HTC, em geral, e BoT, em particular, sobre uma infraestrutura computacional distribuída depende diretamente da escala que a mesma permite. O tamanho do *pool* de processamento, definido como o número de processadores alocados, é o principal promotor de desempenho, enquanto que o esforço de coordenação envolvido é o principal fator de limitação. Para atingir uma vazão extremamente alta é necessário operar eficientemente em escala extremamente alta, assumindo que a distribuição de tarefas para os processadores disponíveis e o fornecimento de qualquer dado de entrada necessário ou coleta dos resultados gerados não sejam um gargalo.

De fato, a execução eficiente de aplicações BoT tem sido relatada em uma variedade de infraestruturas para computação de alta vazão (HTC), que vão desde grades P2P [Litzkow, Livny e Mutka 1988; Cirne et al. 2006] até sistemas massivos de computação voluntária [Anderson et al. 2002; Anderson 2004].

O paradigma de grades de *desktops* (*desktop grids*) já se consagrou como um ambiente apropriado para computação de alta vazão. O Projeto Condor [Litzkow, Livny e Mutka 1988] é reconhecido como o melhor representante existente de tecnologias para dar suporte a grades de *desktops* de alta vazão. Outros sistemas que seguiram a filosofia do Condor provaram também ser igualmente eficazes [Cirne et al. 2006; Oliveira, Lopes e Silva 2002]. Estas infraestruturas genéricas são, entretanto, sistemas de escala limitada. Mesmo se algum tipo de mecanismo de incentivo for usado [Andrade et al. 2007], é improvável que um sistema que integra mais do que algumas dezenas de milhares de computadores possa ser montado. De fato, os maiores sistemas existentes que usam estas tecnologias não possuem mais do que alguns poucos milhares de computadores [Thain, Tannenbaum e Livny 2006].

Plataformas para computação voluntária (*Voluntary Computing*) [Anderson et al. 2002; Anderson 2004], por outro lado, já provaram a sua adequação para prover HTC e podem congregar quantidades enormes de recursos para processar a carga extremamente alta de suas aplicações típicas. Estas infraestruturas poderosas são, entretanto, menos flexíveis em relação aos tipos de aplicações que suportam. Primeiro, porque configurar uma infraestrutura de computação voluntária tem um custo significativamente mais elevado do que executar

aplicações BoT de ciclos de vida curtos sobre grades de *desktops* - isto se deve, principalmente, pelo fato de que é necessário conseguir voluntários para a iniciativa. Desta forma, tais plataformas tendem a ser mais apropriadas para executar aplicações BoT de longa duração cuja carga de trabalho é virtualmente infinita [Anderson et al. 2002]. Além disso, a eficácia da obtenção de recursos voluntários para tais plataformas é profundamente influenciada pelo impacto percebido da aplicação que irá ser executada sobre elas. Em conseqüência, somente algumas aplicações de forte apelo popular podem beneficiar-se da vazão extremamente alta que os sistemas de computação voluntária podem entregar. Mesmo assim, isso só pode ser alcançado se um esforço significativo for dedicado a convencer os participantes voluntários a aderir ao sistema o que, por sua vez, depende, em maior ou menor grau, de fatores tais como o mérito e o apelo público da aplicação, da quantidade de cobertura da mídia recebida, de campanhas de publicidade explícita em meios populares de comunicação, de marketing viral, dos incentivos para os voluntários e de outras atividades de relações públicas [Shiers 2010]. A escalabilidade na implantação deste tipo de projeto também depende de tornar a tarefa de instalação extremamente simples e contar com o proprietário do recurso envolvido ativamente na configuração do sistema. Normalmente, a implantação é bem simplificada, constando basicamente do *download* e da instalação de um programa, o que pode ser facilmente realizado pelo proprietário do recurso. Entretanto, não há uma padronização do que deve ser instalado por cada projeto de computação voluntária, o que requer a repetição do esforço de instalação por parte do voluntário. Por exemplo, um usuário que deseja doar recursos computacionais para os projetos SETI@home [Anderson et al. 2002] ou FightAIDS@home [Scripps 2011] deve instalar duas aplicações específicas e diferentes, cada uma com os seus próprios protocolos e parâmetros.

Se por um lado, o envolvimento do usuário permite a implantação potencial em milhões de recursos com baixo custo, do outro lado, isto torna o crescimento da infraestrutura lento e fora do controle do gestor do projeto de computação voluntária. Além disso, as mudanças no *software* instalado nos recursos são mais difíceis de serem realizadas, a menos que algum procedimento de atualização automática seja fornecido. Isto, por sua vez, pode aumentar as preocupações de segurança por parte dos voluntários e, eventualmente, afetar negativamente a sua vontade de aderir ao sistema. Além disso, a singularidade intrínseca de cada aplicação e a necessidade de configuração inicial, diminui consideravelmente a flexibilidade destas pla-

taformas. Uma vez que um recurso está configurado para suportar um projeto de computação voluntária específico, não pode ser compartilhado com outras iniciativas semelhantes, a menos que ações explícitas dos voluntários sejam tomadas. Note que isso é verdade mesmo para as plataformas que suportam múltiplos projetos, como o BOINC [Anderson 2004], onde o voluntário deve, explicitamente, vincular os projetos desejados (ou todos eles) para a sua identificação e determinar quais recursos ele deseja compartilhar com cada projeto [Shiers 2010].

3.3 O Desafio dos Custos

Para atingir uma vazão extremamente alta, é necessário operar eficientemente em escala extremamente alta. E, como discutido no Capítulo 2, uma das causas da limitação em escalabilidade e elasticidade está relacionada com os custos, diretos e indiretos, para montagem e manutenção do estoque de recursos.

Existe uma expectativa de que os fornecedores de nuvens públicas podem oferecer serviços a preços competitivos e ainda obter lucro. No entanto, a construção de infraestruturas de computação na nuvem exige enormes investimentos iniciais e envolve altos custos operacionais. O estudo de Greenberg *et al.* [Greenberg et al. 2008] mostra que os custos típicos associados com a construção de centros de processamento de dados para nuvens possuem a seguinte distribuição: aquisição de servidores, incluindo *hardware* e *software*, respondem por 45% do custo total; montagem da infraestrutura, incluindo refrigeração e instalações lógicas e elétricas, consomem 25% dos recursos; equipamentos e canais de comunicação em geral são responsáveis por 15% do orçamento e os 15% restantes ficam por conta de fornecimento de energia e outras despesas.

Adicionalmente, Li *et al.* apresentam uma discussão mais detalhada sobre os custos envolvidos com a propriedade e gestão de centros de dados em nuvem e como eles compõem o custo total de propriedade associado (TCO do inglês *Total Cost of Ownership*) [Mieritz e Kirwin 2005]. Na abordagem de Li *et al.* [Li et al. 2009], os quatro principais grupos de custos acima mencionados são expandidos em um arcabouço com oito classificações que, além dos investimentos iniciais, também incluem os custos relacionados com o funcionamento do centro de dados. As oito categorias são: *Servidores*, *Software*, *Rede e Comunicação*, *Suporte*

e Manutenção, *Energia*, *Refrigeração*, *Instalações* e *Custos Imobiliários*. O TCO final do centro de dados é obtido através da soma destes oito componentes de custos.

Além de TCO, que aborda o custo do centro de dados propriamente dito, também é considerado no arcabouço proposto por Li *et al.* o Custo de Utilização (ou UC, do inglês *Utilization Cost*), que corresponde ao custo associado apenas com os recursos sendo efetivamente utilizados pelos clientes, levando em conta a utilização elástica que é suportada. Considerando a virtualização como um padrão entre os provedores, o arcabouço assume que uma máquina virtual (VM) é a unidade básica de consumo em centros de dados de nuvens e propõe a métrica Densidade de VM (do inglês *VM Density*), a qual representa a quantidade de máquinas virtuais suportada por cada servidor físico. Assim, o custo da quantidade total de VMs potenciais (*TVM* = *VM Density* x *qtd servidores físicos*) é independente do nível de uso da estrutura e está incluído no TCO, enquanto que o custo associado com as VMs realmente em uso (variando de 0 até *TVM*) é capturado pelo UC.

Em situações de alta ociosidade no centro de dados, o UC pode não ser representativo do TCO real. A faixa estimada de utilização para servidores convencionais é entre 5 e 20% [Armbrust et al. 2009]. Este baixo nível médio de utilização da CPU foi apurado através de um estudo realizado com 5.000 servidores por seis meses [Barroso e Hölzle 2007]. Com a adoção da virtualização, a utilização média pode chegar a 35% (38% no caso da Google) [Stanoevska-Slabeva e Wozniak 2010]. No caso de provedores de nuvens, há pouca informação disponível sobre o nível de utilização, mas estima-se que a Amazon possuía 40.000 servidores em agosto de 2009 com o alvo de atingir 75% de utilização [CloudScaling 2009]. Por outro lado, a ociosidade potencial em servidores virtualizados pode ser de 65% em centros de dados privados.

Uma característica especial do arcabouço de Li *et al.* é a utilização de um parâmetro da taxa amortizável (*amortizable rate parameter*), obtido através da aplicação de um período de depreciação e do custo do dinheiro sobre os valores de cada investimento ou despesa de forma que os custos possam ser referenciados em intervalos de tempo pequenos como, por exemplo, uma hora de uso. A amortização do TCO de centros de dados de nuvens deve ser feita com o produto da venda dos recursos virtualizados. Desta forma, as VMs que estiverem em uso em um servidor durante um período de tempo específico devem amortizar os custos de todas as VMs suportadas pelo mesmo servidor para o mesmo período de tempo (*VM*

Density). Assim, sempre existirá um ponto de equilíbrio no qual a quantidade de VMs que estão em uso cobrem integralmente os custos totais. Acima deste ponto, o provedor estará operando de forma lucrativa. Neste caso, as VMs não usadas representam a disponibilidade de estoque da nuvem, uma vez que representa o produto efetivamente comercializado pelo provedor - a sua venda (ou não) impacta diretamente nos resultados do negócio e na sua própria amortização.

Estes investimentos iniciais para a montagem de centros de dados para nuvens precisam ser amortizados durante uma vida útil razoável de cada tipo de bem e considerando também o custo do dinheiro. Há uma crescente busca, tanto no mercado como na academia, por alternativas de diminuição do TCO de centros de dados para computação na nuvem, motivados tanto pelos aspectos financeiros em si quanto por questões relacionadas com a relevante pegada (*footprint)* ambiental que as grandes estruturas centralizadas associadas com *cloud computing* têm apresentado. Há diversos desafios envolvidos com cada tipo de custo [Greenberg et al. 2008; Patel e Shah 2005]:

- *Servidores*: Os riscos inerentes ao planejamento de capacidade de centros de dados para nuvens pressionam os custos para cima. A necessidade de atender às necessidades dos clientes e respeitar os SLA contratados frequentemente leva a um dimensionamento desigual entre demanda e capacidade. A incerteza em prognósticos de utilização e a necessidade de planejamento a longo prazo, para acomodar prazos de entrega de fornecedores, também induzem à um gerenciamento de risco.

- *Rede*: Os investimentos em *switches*, roteadores, balanceadores de carga e outros equipamentos representam uma parte significativa dos custos com redes em centros de dados. Entretanto, os custos para comunicação usuário-centro de dados e centros de dados-centros de dados (*wide area networking*) são também muito relevantes e suscetíveis à influência de uma série de aspectos como dinâmica do mercado, tarifação, tráfego etc. É necessário equilibrar os custos e, ao mesmo, garantir uma latência de resposta adequada para os clientes.

- *Energia*: O alto preço da energia e a tendência de uso sustentável dos recursos ambientais pressionam para que ocorra uma diminuição do consumo de energia em centros de dados. Entretanto, aspectos como uso ineficiente de energia pelo *hardware*, per-

das na distribuição e a gasto adicional de energia para dissipar o calor gerado são obstáculos que precisam ser contornados ainda. Métricas recentes como eficiência no uso energético em centros de dados (PUE do inglês *Power Usage Efficiency*) [GreenGrid 2010] e proporcionalidade de energia em servidores (EP do inglês *Energy Proportionality* [Barroso e Hölzle 2007] começam a ser adotadas e espera-se também o surgimento de inovações que impactem no consumo dos servidores e ajudem a reduzir o custo total de energia dos centros de dados.

- *Infraestrutura*: Correntemente, os custos com infraestrutura representam um dos mais relevantes *overheads* dos centros de dados para nuvens. A grande concentração de servidores em enormes centros de dados exige um proporcional investimento em recursos dedicados tanto para a distribuição consistente de energia quanto para a consequente dissipação do calor produzido. São necessários geradores, transformadores, condicionadores de ar e UPS (do inglês *Uninterrupted Power Supply*) de larga escala que não são produzidos em série, exigindo pedidos por encomenda de alto custo e grande prazo de entrega. Além de dificultar o planejamento, tais equipamentos ainda demandam um grande tempo para amortização (cerca de 15 anos).

Tanto as infraestruturas para a montagem de nuvens privadas quanto aquelas usadas em nuvens públicas compartilham as mesmas preocupações com relação aos custos de montagem e funcionamento de centros de dados. Desta forma, para as empresas que são elegíveis para manter as suas próprias nuvens privadas, o custo para a utilização de recursos equivalentes em uma nuvem pública tende a ser mais caro ao longo do tempo, pois a última opção também incorpora no preço cobrado, além dos custos comuns, o lucro do provedor, os riscos envolvidos com o provisionamento de recursos e com o atendimentos de SLAs.

Entre as propostas para reduzir os custos dos centros de dados em nuvem que começam a surgir [Greenberg et al. 2008; Patel e Shah 2005; GreenGrid 2010; Barroso e Hölzle 2007], podemos citar:

- *harmonização e melhor posicionamento entre as abordagens de super centros de dados e micro centros de dados* [Barroso e Hölzle 2007]: Esta proposta baseia-se na harmonização entre localização e tamanho de centros de dados, e considera o uso combinado de dois tipos de infraestruturas: os chamados *Mega DC*, com dezenas de

milhares ou mais servidores, com custos de implantação que podem atingir 2 bilhões de doláres e consumo de energia na casa dos 20 MW; e os *Micro DC*, com cerca de mil servidores em média que são acondicionados em um *container*, custam cerca de 2 milhões de doláres e demandam 500 KW de energia. Cada uma das abordagens apresenta vantagens específicas que tornam-se mais ou menos relevantes de acordo com o cenário considerado. Os benefícios da economia de escala continuam sendo a principal vantagem de adoção de *Mega DCs* para computação na nuvem, considerando que as tecnologias de virtualização e o alto grau de automação atingido potencializam o compartilhamento de recursos e custos. No caso dos *Micro DCs*, destacam-se os menores custos e prazos para implantação e maior eficiência de comunicação, em termos de velocidade e latência, proporcionada pela possibilidade de instalação em áreas mais próximas do cliente. Tendo em conta o particionamento e replicação de dados, são ainda necessários métodos adequados para o projeto e gestão de tráfego em toda a rede de *Micro* e *Mega DCs*, bem como melhores mecanismos para mapear usuários para centros de dados;

- *agilidade da estrutura de rede para aumentar e diminuir dinamicamente os recursos em função da demanda* [Al-Fares, Loukissas e Vahdat 2008]: A *agilidade* em um centro de dados pode ser descrita como a possibilidade de que qualquer serviço pode ser alocado dinamicamente para qualquer servidor em qualquer lugar do centro de dados, mantendo a segurança adequada e o isolamento de desempenho entre os serviços. Neste sentido, a rede interna se destaca como uma barreira na agilidade e aumenta a fragmentação de recursos que leva à diminuição do nível de utilização por servidor. Várias abordagens estão sendo exploradas para atender melhor aos requisitos de redes internas dos centros de dados para nuvens. Em particular, para melhorar a capacidade de aumentar e diminuir dinamicamente os recursos para atender a demanda e alocar esses recursos para clientes e serviços considerando a localização ideal dentro do centro de dados;

- *resiliência em nível de micro centros de dados geograficamente distribuídos* (do inglês *geo-diverse micro data centers*) [Greenberg et al. 2008]: Partindo do princípio que a resiliência seja mantida em nível do centro de dados, esta abordagem considera que

as camadas de redundância dentro de cada centro de dados podem ser retiradas. Isto seria obtido através da instalação distribuída geograficamente de vários *Micro DC* sem geradores de energia ou UPS atuando como espelhos uns dos outros. Esta proposta apresenta potencial para fornecer um grau relativamente elevado de independência entre falhas físicas dos centros de dados (por exemplo, falta de energia), e uma oportunidade para atingir os clientes de cada centro de dados com menor latência. Entretanto, há ainda problemas em aberto, incluindo o desenvolvimento de estratégias adequadas para obter o equilíbrio entre o grau de resiliência ainda necessária dentro de cada centro de dados com relação à resiliência obtida em nível de centros de dados espelhados, bem como o impacto da adoçao de cada estratégia sobre as aplicações;

- *aumentar a taxa de utilização da infraestrutura* [Stanoevska-Slabeva e Wozniak 2010]: Os servidores devem estar envolvidos na produção de receitas. Considerando que há custos fixos para cada servidor instalado em um centro de dados e que o tempo de vida de um servidor é de cerca de três anos, é fundamental para o provedor de serviços que todos os servidores estejam operantes e envolvidos em atividades que produzam receita e maximizem os investimentos realizados. O desafio é conseguir eficiência na distribuição da demanda sobre os recursos disponíveis para manter sob controle o crescimento da infraestrutura. Uma forma de se obter isto é garantir que qualquer servidor possa ser aplicado a qualquer demanda para permitir a concentração da ociosidade da infraestrutura em um grupo de servidores totalmente disponíveis que pode ser mantido em um tamanho controlado.

Mecanismos mais elaborados para aumentar o nível de utilização dos servidores através do uso de modelos de precificação específicos começaram a surgir para a computação na nuvem, de modo a conciliar uma maneira de usar o excesso de estoque criado sem comprometer o nível de serviço dos prestadores. Uma iniciativa criativa para explorar a eventual ociosidade em seus centros de dados foi lançada pela *Amazon Web Services* (AWS) [Amazon 2010] recentemente. Juntando-se às duas opções já existentes: *on-demand instance* e *reservation instance*, a *spot instance* [Amazon 2011] é a terceira alternativa de preços para o serviço AWS EC2. No melhor estilo da lei de oferta e demanda, a opção *spot instance* permite que os usuários ofereçam um preço pela capacidade não utilizada da infraestrutura

da AWS, o *bid price*. A AWS, por sua vez, determina o *spot price*, um valor dinâmico para eventuais recursos ociosos com base na utilização dos seus centros de dados. A instância do usuário executa enquanto o seu *bid price* for maior do que o *spot price* e pode ser interrompida a qualquer momento. Neste caso, a AWS não oferece nenhuma garantia, além do fato de que o usuário não será cobrado por qualquer hora parcial que sua instância tenha consumido desde que foi terminada pela AWS. O site da AWS recomenda *spot instances* para clientes com flexibilidade com relação ao momento em que suas aplicações podem ser executadas e para as aplicações cuja arquitetura permita fazer progressos, mesmo que o processamento seja interrompido (por exemplo, adicionando pontos de controle e dividindo o trabalho em pequenas unidades).

Colocar as *spot instances* da AWS em perspectiva nos induz a duas conclusões:

1. a existência de ociosidade em infraestruturas computacionais continua a ser um aspecto recorrente na maioria dos paradigmas e abordagens;

2. as aplicações têm necessidades diferentes e há demanda por infraestruturas computacionais com baixos níveis de QoS, mas que sejam atrativas economicamente.

No próximo capítulo apresentaremos uma categoria diferente de recursos computacionais que podem ser usados no provimento de serviços computacionais de alta escalabilidade e elasticidade: aqueles que pertencem a terceiros.

Capítulo 4

Provisão de Computação na Nuvem usando Recursos Terceirizados

Neste capítulo, nós abordamos o problema de planejamento de capacidade para o provisionamento de centros de dados para computação na nuvem e propomos o uso de recursos terceirizados para tal finalidade.

O restante do capítulo está organizado da seguinte forma. Na Seção 4.1 é feito um esboço da abordagem para provisão de infraestruturas computacionais usando recursos terceirizados. A seguir, na Seção 4.2, nós apresentamos a categoria de recursos terceirizados de baixa escala. Na Seção 4.3, nós apresentamos o conceito de *Just in Time Clouds*, uma abordagem alternativa, baseada em recursos terceirizados, para a montagem de infraestruturas computacionais para suporte à computação na nuvem, chamadas *JiT Data Centers* ou JiT DCs. Finalmente, na Seção 4.4, nós apresentamos as nossas considerações finais.

4.1 Esboço da Solução

Apesar das facilidades e vantagens oferecidas pelo paradigma de computação em nuvem, já discutidas anteriormente, ainda existem obstáculos à sua adoção por parte de algumas empresas e instituições, pelo menos no curto prazo [Golden 2009]. A falta de uma padronização de APIs (do inglês *application programming interfaces*) para o provisionamento de serviços, dificuldades em adaptar as aplicações para a arquitetura adotada pelo provedor selecionado, níveis de segurança, privacidade e controle inadequados para alguns segmentos, existência

de riscos estratégicos e comerciais ainda não completamente cobertos pelos acordos de nível de serviços oferecidos e restrições legais ou regulatórias são algumas das principais causas que impedem que esses clientes potenciais utilizem os serviços oferecidos por provedores de computação em nuvem.

Naturalmente, alguns destes clientes potenciais podem ainda se beneficiar do paradigma de computação na nuvem através da adoção das mesmas tecnologias e estratégias utilizadas pelos provedores de computação em nuvem, a fim de reduzir o TCO de suas infraestruturas de TI próprias. Isto é particularmente adequado para os clientes com uma infraestrutura de TI de grande porte que podem se beneficiar de economias de escala semelhantes. No entanto, não importando se tais clientes potenciais usam uma abordagem de nuvem privada[1] ou não, eles continuam a manter seus recursos próprios de computação e precisam fazer planejamento de capacidade, normalmente tendo que arcar com o ônus de manter recursos em excesso para suportar picos de sua demanda. Isto implica na existência de recursos excedentes com relação à operação padrão do negócio e que, eventualmente, ficam ociosos.

Considerando uma gradação dos detentores de recursos computacionais terceirizados excedentes do ponto de vista da escala, ou seja, pela quantidade de recursos excedentes disponíveis, podemos considerar que existe um ponto de corte da magnitude que os separa em dois grupos. O primeiro grupo é dos que ficam acima do ponto de corte e possuem capacidade excedente suficiente para poderem atuar como provedores públicos de computação na nuvem, oferecendo os seus recursos excedentes para outros, como fez a Amazon Bookstore, por exemplo. Abaixo do ponto de corte, situam-se todos aqueles que não possuem, sozinhos, recursos terceirizados excedentes suficientes para uma atuação solo. O espectro de escala imediatamente inferior ao ponto de corte engloba recursos pertencentes a instituições ou a indivíduos, incluindo desde empresas de grande porte, passando por centros de dados de pequeno e médio porte até chegar ao menor nível de agrupamento, servidores e recursos individuais, convencionais ou não convencionais. Neste trabalho, nós estamos especialmente interessados nesta última categoria, que chamamos de *recursos terceirizados de baixa escala*.

Os recursos terceirizados de baixa escala que consideramos podem estar, eventualmente,

[1]Conforme visto no Capítulo 3, o termo *nuvem privada*, em oposição a infraestruturas públicas operadas por provedores de computação na nuvem, tem sido usado para descrever este tipo de infraestrutura.

dispersos e serem mantidos (ou, pelo menos, operados) por um grande número de indivíduos e/ou organizações diferentes. Organizados em uma cadeia de produção baseada na filosofia "*Just in Time*", os detentores de recursos terceirizados poderiam ser federados para atuar como fornecedores de um tipo particular de centros de dados em nuvem, que chamamos *JiT Data Centers* ou *JiT DCs*. Estes centros de dados podem ser montados pelos fornecedores somente quando solicitado pelos clientes e exatamente nas condições exigidas. Note-se que o que estamos propondo não é semelhante a outros provedores especializados de nuvens que constroem os seus serviços em cima de outros fornecedores de IaaS e, portanto, não precisam implantar infraestrutura própria (ex. rightscale.com [Rightscale 2011]). O serviço que um provedor de nuvem baseado em recursos terceirizados oferece é exatamente o mesmo fornecido pelos provedores tradicionais de nuvens públicas, portanto, não faz sentido comprar serviço a partir do último e vender o mesmo serviço, sem acrescentar qualquer valor a ele. O diferencial é que através da descoberta, recuperação e revenda de recursos tercerizados excedentes, um provedor interveniente de tais recursos também é capaz de operar sob a filosofia *Just in Time* para permitir que grandes quantidades de recursos possam ser contratados por um único cliente por um período de tempo relativamente curto e depois liberados.

4.2 Recursos Terceirizados de Baixa Escala

Nossa abordagem considera que parte dos recursos computacionais utilizados para apoiar as operações de vários negócios se enquadram na categoria de recursos terceirizados excedentes, representando uma capacidade provisionada e disponível para períodos de alta demanda, mas permanecendo inativa durante parte do tempo. Para esses recursos já implantados e em operação, qualquer possibilidade de utilização em momentos de ociosidade, mesmo que para uma finalidade diferente daquela originalmente especificada, pode levar a um lucro adicional ou pelo menos para a redução do seu TCO.

Um primeiro passo para a possível utilização de recursos terceirizados ociosos é o dimensionamento dos recursos excedentes, ou seja, a capacidade ociosa real disponível. O cálculo do excedente potencial deve levar em consideração a demanda histórica de pico para curto e longo prazo que permite a criação de uma margem de segurança confortável para a operação do negócio original. Seja $\boldsymbol{C}$ a capacidade total de recursos computacionais instala-

dos no ambiente ***E*** para suportar o negócio ***B***. O valor apropriado para ***C*** é obtido por meio de planejamento da capacidade que considera as necessidades operacionais e estratégicas do negócio durante um determinado período de tempo.

O nível de utilização de ***E*** é a fração de ***C*** consumida pela operação do negócio ***B***, referida como ***u***. Devido à dinâmica específica de cada contexto, ***u*** pode variar dependendo do tempo e u_t representa a utilização máxima (*anticipated peak load*) [Simmons, McCloskey e Lutfiyya 2007] de ***C*** no tempo ***t***.

O excedente ocioso ***S*** sobre ***E*** no momento ***t***, denotado como $\mathbf{S_t}$, é obtido pela aplicação em ***C*** do complemento da taxa real de utilização em ***t***:

$$S_t = C \times (1 - u_t) \tag{4.1}$$

Assim, S_t é a fração da capacidade ***C*** existente no ambiente ***E*** que está ociosa no momento ***t*** e pode ser usado por uma duração específica para outros fins que não ***B***. Este relacionamento é ilustrado a seguir na Figura 4.1.

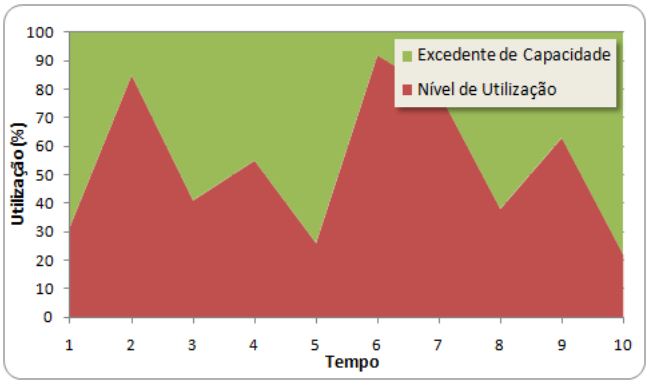


Figura 4.1: Excedente de Recursos Terceirizados

Neste trabalho, nós estamos nos concentrando em contextos onde a quantidade de recursos terceirizados excedentes disponíveis (S_t) não alcança uma magnitude ***M*** que permite que os seus proprietários sozinhos possam atuar como um provedor público de computação na nuvem, i.e. eles são *recursos terceirizados de baixa escala*. Nas seções seguintes, nós apresentamos uma abordagem em que um provedor age como um agente de ligação para permitir que diferentes contextos com recursos terceirizados de baixa escala possam oferecer, em conjunto e de forma federada, nuvens públicas de magnitude maior ou igual a ***M***.

4.3 *Just in Time Clouds*

Nossa proposta apresenta uma abordagem alternativa para construir infraestruturas computacionais para suporte à computação na nuvem que não é baseado em planejamento de capacidade tradicional. Inspirados na filosofia "*Just in Time*" (JiT) da Toyota [Toyota Motor Co 2011], nós introduzimos o conceito de *Just in Time Clouds* para representar uma nova categoria de serviço na qual o provedor apenas aloca recursos quando efetivamente demandados pelos clientes e somente enquanto houver uso para eles.

Dessa forma, é esperado que um provedor de uma *JiT Cloud* seja capaz de oferecer computação na nuvem de forma muito mais elástica, posto que baseia-se na descoberta e revenda de recursos terceirizados de baixa escala de uma federação de fornecedores. O custo de coordenação da federação é o insumo mais relevante para o *JiT Provider*, pois o ônus do custo de disponibilidade (e eventual ociosidade) dos recursos permanece como uma responsabilidade dos seus proprietários e o custo de utilização somente ocorre quando os recursos são efetivamente utilizados.

4.3.1 *JiT Providers* e *JiT Data Centers* (JiT DCs)

Em nossa abordagem, o *Just in Time Provider* é um provedor de computação em nuvem pública que, em vez de montar e manter uma estrutura própria de centros de dados para operar o seu serviço, faz uso de uma federação de recursos terceirizados de baixa escala já existentes em contextos privados. Ao contrário de intermediários de fornecedores convencionais de computação na nuvem, um *Just in Time Provider* não representa nenhum provedor público de computação na nuvem, mas age como um provedor legítimo e totalmente autônomo, que tira proveito de recursos que poderiam estar irremediavelmente subutilizados sem sua intervenção.

Um *JiT Provider* agrega valor pela oferta de computação na nuvem sem a necessidade de lidar com planejamento de capacidade tradicional, mas simplesmente descobrindo, preparando e revendendo recursos tercerizados excedentes. A escalabilidade e a elasticidade ficam limitadas apenas pela capacidade do *JiT Provider* em montar uma cadeia de fornecimento de recursos terceirizados grande o bastante.

Os recursos a serem operados pelo *JiT Provider* podem vir de fontes tão diversas como

um único proprietário de um centro de dados virtualizado com excesso de capacidade mantido para suportar demandas de pico de seu próprio negócio (como especula-se que foi a motivação para o surgimento da AWS), quanto de usuários de uma rede de TV digital federados pela emissora, que franqueiam o uso de seus receptores (*set-top-boxes*) [Batista et al. 2007].

Cada conjunto (*pool*) de recursos terceirizados excedentes existente em um determinado ambiente representa uma abstração chamada *Just in Time Data Center* (*JiT DC*). Cada *JiT DC* reúne uma certa quantidade de recursos com determinadas características e capacidades, chamados *JiT Resources*, que devem ser identificados e classificados pelo *JiT Provider*. Dependendo do seu tipo, um *JiT Resource* pode ser adequadamente especializado como, por exemplo, uma *JiT VM* para representar uma máquina virtual específica dentro de um *JiT DC* específico. Entre as diversas características gerais de um *JiT DC*, estão o nível de serviço suportado por seus recursos e as condições negociadas (ou arbitradas) pelo proprietário para o seu uso. Uma *Just in Time Cloud* (Figura 4.2) consiste de um conjunto de *JiT DCs* incorporados e coordenados pelo *JiT Provider* para a provisão de serviços públicos de computação na nuvem.

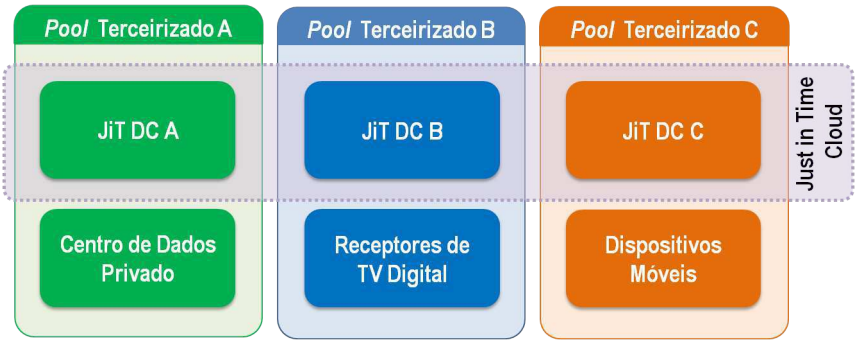


Figura 4.2: Composição de de uma *JiT Cloud*

Os *JiT Resources* que são integrados em *JiT Data Centers* podem ser classificados em **dedicados**, quando estão totalmente alocados para uso pelo *JiT Provider* por um certo período de tempo, e **não dedicados**, quando sua atribuição é parcial, sendo compartilhado de forma oportunista, e com a possibilidade de serem retomados por seus proprietários correspondentes sem qualquer aviso prévio. No primeiro caso, existe a reserva e níveis de disponibilidade negociados antecipadamente. No segundo caso, os recursos são voláteis e podem sofrer falhas ou retomada a qualquer momento. Em ambos os casos, o *JiT Provider* precisa lidar com

questões de eventuais migrações e levar em conta o tempo necessário para alocar e desalocar os recursos.

Um dos principais requisitos arquiteturais para suportar *Just in Time Clouds* diz respeito ao particionamento adequado dos recursos terceirizados entre a operação prioritária do negócio principal do proprietário dos recursos, quando for o caso, e o aproveitamento da capacidade eventualmente ociosa pelo *JiT Provider*. Esta coexistência, na prática, significa a manutenção de dois *pools* lógicos de recursos construídos sobre os mesmos recursos físicos. O primeiro *pool* lógico é constituído pelos recursos em uso efetivo (u_t) acrescido de uma margem de segurança. Este *pool*, que chamaremos de *Private DC*, é integralmente gerenciado pelo proprietário dos recursos terceirizados, garantindo os aspectos estratégicos e operacionais do seu negócio original. O segundo *pool* representa o *JiT DC* propriamente dito e é constituído pelos recursos de *C* remanescentes (S_t). Devido ao caráter prioritário da operação mantida pelo *Private DC* e a definição altamente dinâmica dos recursos disponíveis para o *JiT DC*, são necessários mecanismos eficientes para coordenar a migração de recursos entre os dois *pools* sempre que requisitados ou liberados pelo *Private DC*.

Essa segregação pode ser totalmente suportada pelas tecnologias disponíveis atualmente e a dinâmica para a transição de recursos entre os dois *pools* pode ser operacionalizada através de mecanismos de priorização.

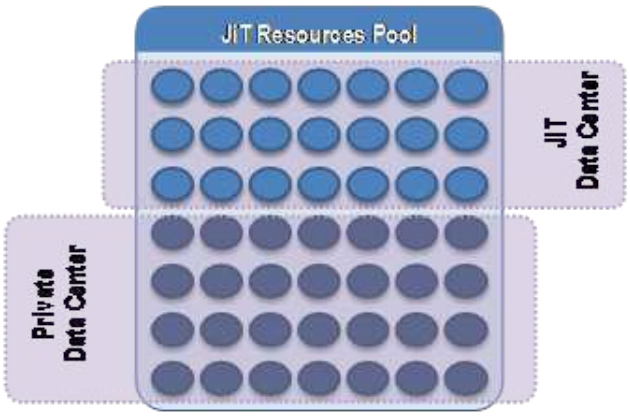


Figura 4.3: Representação da separação de *Private DC* e *JiT DC* sobre um *pool* de recursos terceirizados

A seguir, será feita uma breve discussão de como os recursos terceirizados podem ser classificados com relação a algumas de suas características. Em especial, serão focadas as

singularidades que podem impactar no seu aproveitamento em *JiT Clouds*.

4.3.2 Padrões de Granularidade, Volatilidade e Dispersão de Recursos Terceirizados

As *JiT Clouds* podem ser montadas sobre recursos que estejam distribuídos por todo o espectro de recursos terceirizados de baixa escala. Uma das missões do *JiT Provider* é descobrir e explorar o potencial dos recursos disponíveis alinhando-os com as necessidades das aplicações de clientes. Dependendo de suas características, os recursos terceirizados podem fornecer diferentes níveis de qualidade de serviço, elasticidade e escalabilidade. O nível de qualidade de serviço oferecido por um *JiT DC* é totalmente dependente do nível de qualidade de serviço suportado pelos recursos usados para montá-lo, o qual está relacionado ao padrão de granularidade, volatilidade e dispersão dos recursos.

Por granularidade [wiseGEEK 2012], entende-se o nível de fragmentação da capacidade computacional provida por cada recurso terceirizado. Nesta classificação, servidores de alta capacidade e *clusters*, representam **recursos terceirizados de baixa granularidade** (*coarse-grained*), que são mais densos e poderosos, enquanto que computadores pessoais representam **recursos terceirizados de alta granularidade** (*fine-grained*), mais leves e de menor capacidade, sendo necessário diminuir o tamanho da tarefa (ou "grão") a ser processada nos mesmos.

Volatidade, por sua vez, representa o nível de disponibilidade e confiabilidade que o recurso terceirizado oferece quando alocado para uma determinada tarefa. Dedicação exclusiva, mecanismos de contingenciamento e tolerância a falhas caracterizam os **recursos terceirizados de baixa volatilidade**, enquanto que o uso oportunista e a falta de garantias de funcionamento são as principais características dos **recursos terceirizados de alta volatilidade**.

A última propriedade considerada, a dispersão, está relacionada com o nível de distribuição dos recursos terceirizados. Os recursos concentrados em centros de dados representam **recursos terceirizados de baixa dispersão** enquanto que recursos individuais, distribuídos geograficamente, são **recursos terceirizados de alta dispersão**.

Quando os recursos estão concentrados em centros de dados e sua capacidade está locali-

zada mais próxima do topo da magnitude que limita a baixa escala de recursos terceirizados, os níveis de serviço oferecidos são consistentes com os praticados pelos provedores tradicionais de computação na nuvem. Dessa forma, *JiT DCs* baseados em recursos de baixa granularidade, baixa volatilidade e baixa dispersão podem ser usados para hospedar quaisquer das aplicações tipicamente suportadas por computação na nuvem.

No outro extremo do espectro da escala, quando os recursos terceirizados são de grão pequeno e distribuídos, eles precisam ser agrupados e coordenados pelo *JiT Provider* para a sua exploração. Estes recursos de alta granularidade, alta volatilidade e alta dispersão podem ser **convencionais**, representados por equipamentos padrão de processamento, e **não convencionais**, incluindo *tablets*, *PDAs*, telefones celulares e receptores de TV Digital. Todos esses dispositivos não convencionais são equipados com processadores poderosos e quantidade razoável de memória, permitindo-lhes a execução de aplicações. No entanto, como estes dispositivos são tipicamente recursos não dedicados e voláteis, um *JiT DC* baseado neles é, possivelmente, menos confiável do que aquele que é construído sobre recursos privados e dedicados. No entanto, existem evidências suficientes de que existem clientes dispostos a utilizar tais serviços *best-effort*: por um lado, a mera existência das *spot instances* da AWS é uma boa indicação disso; por outro lado, a abundância de aplicações HTC científicas e industriais, susceptíveis de serem executadas em ambientes de nuvem com qualidade de serviço equivalente ao proporcionado pelas *spot instances* da AWS, são indicativos adicionais de que um serviço altamente elástico e escalável de computação na nuvem, mesmo quando baseado em tais recursos, é de muita utilidade.

Há vários desafios envolvidos com o uso de recursos com granularidade muito alta e de alta dispersão para construir *JiT DCs*. O fracasso de companhias (e.g. Distributed.net [May 1999]) que tentaram vender poder computacional de terceiros (e não doar, como é o caso de iniciativas de computação voluntária como SETI@Home [Anderson et al. 2002] e outros [Stanford 2011] [Scripps 2011]) sugere que há um componente mercadológico que deve ser considerado no uso de grãos muito pequenos. Um dos obstáculos é o custo transacional envolvido na identificação, bilhetagem e remuneração de uma quantidade muito grande de transações relacionadas a um número muito grande de pequenos fornecedores. Além de controlar a remuneração devida para cada fornecedor de recursos, existem os custos operacionais para realizar o pagamento efetivo dos fornecedores que podem, em muitos casos,

superar o valor do pagamento em si. Há também o fato de que os ganhos auferidos pelos proprietários de recursos individuais podem ser muito pequenos ou insignificantes e servir como desestímulo à participação [2].

Mesmo no caso de recursos de baixa granularidade, também há desafios e lacunas atualmente. A falta de padronização e interoperabilidade de aplicações entre ambientes completamente virtualizados representa uma necessidade legítima da comunidade atual de usuários de nuvens [Lee 2010] e é um requisito recorrente para aqueles usuários potenciais que ainda não migraram para tal ambiente por causa de tal limitação [Golden 2009]. Isto envolve tanto aspectos estratégicos (dependência de fornecedores ou *vendor lock-in*, concorrência de mercado etc) quanto aspectos de viabilidade técnica (migração dinâmica de VMs em nuvens híbridas). Como há grandes operadores de serviços públicos competindo pela hegemonia de um mercado em formação, cada um deles procura impor o seu modelo de operação como padrão. Dessa forma isolada, as iniciativas de mercado desenvolvem, mantém e evoluem soluções próprias que estão direcionando o avanço e a consolidação do paradigma de computação na nuvem – havendo ainda uma tímida contribuição da academia neste sentido [Lee 2010]. Entretanto, algumas iniciativas, como *Cloud Standards* [CloudStandards 2011], *Cloud Security Alliance* [Alliance 2011] e *Distributed Management Task Force* [Force 2011], já começam a produzir os primeiros resultados nesta direção como, por exemplo, o padrão *Open Virtualized Format* (OVF) [Force 2011]. Além disso, alternativas de *middleware* de código aberto para computação na nuvem como Eucalyptus [Eucalyptus 2011], OpenNebula [OpenNebula 2011]) e o mais recente OpenStack [OpenStack 2011] emergem com facilidades de integração e começam a ser utilizados largamente. À medida que a força deste movimento cresce, espera-se que deva provocar alguma reação dos principais provedores comercias em direção a uma convergência.

A tendência de virtualização de recursos de forma padronizada, em centros de dados privados ou em provedores comerciais, propiciará as condições ideais para a atuação de *JiT Providers*. Os recursos terceirizados em centros de dados privados, em se confirmando a tendência de uma trajetória privada-híbrida-federada-pública para adoção de nuvens [Lee

[2]Dentre as possibilidades para eventuais trabalhos futuros sugeridas no Capítulo 8, encontra-se a investigação de modelos de negócios baseados do uso de agentes aglutinadores para viabilizar o uso de recursos terceirizados com granularidade muito alta e pertencentes a múltiplos proprietários individuais.

2010], poderão ser utilizados para a composição de *JiT Data Centers* que já operem dentro de padrões estabelecidos de instanciação e migração de recursos.

4.4 Considerações Finais

O conceito de *Just in Time Clouds* proposto aqui pode ser considerado como uma alternativa ao modelo padrão de centros de dados centralizados adotado em nuvens públicas e privadas atualmente. Entretanto, quando se considera a possibilidade do uso de recursos terceirizados heterogêneos, com diferentes configurações e níveis de serviço, algumas suposições correntes para a construção de infraestruturas de nuvens tendem a não ser totalmente aplicáveis. Assim, algumas preocupações que não estão presentes na implantação de centros de dados tradicionais para computação na nuvem precisam ser consideradas na construção e operação de *JiT Data Centers* para a montagem de uma *JiT Cloud*.

Dentre os aspectos que precisam ser considerados, podemos citar:

- Como alocar e controlar, sob demanda, os recursos em cada *JiT DC*?
- Quais mecanismos de provisionamento e relocação de recursos são necessários?
- A eventual sobrecarga do esforço envolvido de controle e coordenação é aceitável?
- Como garantir escalabilidade e disponibilidade para *JiT Clouds* baseadas em recursos heterogêneos?
- Que mecanismos de segurança são mais eficientes?
- Há cenários/tecnologias correntes que podem ser explorados através de *JiT Providers*?
- O potencial computacional de dispositivos não convencionais o tornam adequados para uso em HTC?

Algumas dessas questões serão abordadas nos próximos capítulos para os cenários mais desafiadores, que envolvem recursos terceirizados de alta granularidade, alta volatilidade e alta dispersão.

Neste sentido, nós iremos nos concentrar na investigação da viabilidade de construção de *JiT DCs* usando recursos terceirizados voláteis e distribuídos. Em especial, nós apresentaremos uma nova arquitetura que é capaz de lidar com os requisitos para a construção de *JiT DCs* dinâmicos e elásticos baseados em recursos de alta granularidade e alta volatilidade (Capítulo 5) e também discutiremos como tal arquitetura pode ser aplicada para o aproveitamento de recursos terceirizados não convencionais (Capítulo 6).

Capítulo 5

JiT DCs Baseados em Dispositivos de Alta Granularidade, Alta Volatilidade e Alta Dispersão

A fim de construir *JiT Clouds* dinâmicas e de alta vazão baseadas em recursos terceirizados dispersos, de pequena capacidade e não dedicados é necessário fornecer uma maneira de acessar, individualmente, uma grande quantidade de processadores, enviar programas e, possivelmente, dados, para todos e, remotamente, desencadear a execução do código transmitido. Em seguida, reunir os resultados produzidos, e, finalmente, liberar os recursos alocados de forma que outras aplicações possam usá-los.

A ideia de alocar uma enorme quantidade de recursos através da abstração de um *JiT DC*, habilitá-los para o processamento distribuído de aplicações paralelas (centenas de milhares de computadores conectados via Internet, por exemplo) e fazê-lo a um custo menor do que alternativas tradicionais, apesar de atrativa, representa um desafio não trivial. A questão principal é: **onde** encontrar uma grande quantidade de processadores terceirizados disponíveis e **como** configurá-los em conformidade e instantaneamente para o uso em *JiT Clouds* dinâmicas voltadas para os requisitos de alta vazão de aplicações HTC? Além disso, como executar esta tarefa com um atraso mínimo?

Neste sentido, uma categoria singular de dispositivos tercerizados desperta um interesse especial para este trabalho: aqueles que podem ser organizados em uma rede de *broadcast*[1].

[1]O termo *broadcasting* está, originalmente, relacionado a transmissões de rádio ou televisão e significa a

Uma rede de *broadcast* possui o potencial de permitir a comunicação quase simultânea com todos os dispositivos conectados, os quais podem ser coordenados para realizar uma determinada ação. Nesta abordagem, programas transmitidos através do canal de *broadcast* podem ser carregados e executados concomitantemente por todos os recursos computacionais conectados à rede de *broadcast* em um dado momento. Este mecanismo torna possível construir, de uma forma realmente rápida[2] e controlada, *JiT DCs* distribuídos de alta vazão.

Neste capítulo, nós analisamos o potencial de uso de recursos de alta granularidade, alta volatilidade e alta dispersão, no contexto de redes de *broadcast*, para a composição de *JiT DCs* de alta vazão através do uso de mecanismos específicos para a sua descoberta, alocação e coordenação. Nossos resultados de simulação mostram que, mesmo em cenários de altíssima volatilidade de nós, é possível construir *JiT Clouds* com a disponibilidade coletiva [Andrzejak, Kondo e Anderson 2008] adequada para atingir níveis controlados de vazão computacional.

O resto do capítulo está organizado como segue. Na Seção 5.1, nós discutimos alguns requisitos envolvidos na construção de *JiT DCs* de alta vazão voltados ao processamento de aplicações HTC e como as tecnologias atuais os atendem. Em seguida, nós apresentamos na Seção 5.2 uma arquitetura nova para a construção de infraestruturas computacionais distribuídas (DCI, do inglês *Distributed Computing Infrastrucuture*) dinâmicas baseadas em recursos voláteis e dispersos, organizados em uma rede de *broadcast*. Ainda nessa seção, nós apresentamos o modelo de operação da arquitetura proposta. Como em muitas DCI, as questões de segurança são uma preocupação relevante. Nós discutimos na Seção 5.3 os aspectos de segurança relacionados com a operação de sistemas que seguem a arquitetura proposta e apresentamos um modelo de segurança geral que atende os requisitos de segurança identificados. Outras questões importantes de implementação são discutidas na

distribuição, de forma simultânea e através de um meio físico específico e unidirecional (o canal de *broadcast*), de sinais de áudio e/ou vídeo contendo programação para uma determinada audiência. Considerando o mesmo princípio de transmissão de um-para-muitos, será usado o termo **rede de *broadcast*** para representar uma rede composta por um transmissor digital de dados, um **canal de *broadcast***, um conjunto de equipamentos receptores com capacidade de processamento de aplicações paralelas e possibilidade de acesso a um canal de interação *full-duplex*, comumente uma conexão com a Internet.

[2]Na verdade, o quão rápido o *software* será carregado dependerá do tamanho dos dados a serem transmitidos e da velocidade do canal de *broadcast*.

Seção 5.4. Uma análise preliminar do desempenho do sistema baseada em simulação é realizada na Seção 5.5, que traz uma discussão do modelo de simulação utilizado e dos desafios relacionadas com as características particulares dos *JiT DCs* de alta vazão estudados neste capítulo. Em seguida, é feita uma descrição de como foi realizada a nossa avaliação e uma análise dos resultados obtidos nos nossos experimentos. Finalmente, nós apresentamos as nossas considerações finais na Seção 5.6.

5.1 Requisitos para *JiT DCs* de Alta Vazão

Conforme discutido anteriormente, a vazão obtida por uma aplicacão HTC depende diretamente da escala suportada pela infraestrutura computacional sobre a qual a mesma é executada. Para atingir uma vazão extremamente alta, é necessário operar eficientemente em escala extremamente alta. Em outras palavras, aplicações HTC/BoT podem facilmente se beneficiar da disponibilidade de um *pool* massivo de processadores para incrementar a sua vazão, desde que tenha sido garantida que nem a distribuição de tarefas para os processadores disponíveis nem o fornecimento de qualquer dado de entrada necessário ou coleta dos resultados gerados representem um gargalo.

O uso eficiente de recursos terceirizados por aplicações HTC com tarefas de curta duração (*short-lived*) requer a capacidade do *JiT DC* de alta vazão de instanciar um grande *pool* de recursos (ou instância DCI) para uma aplicação a qualquer tempo e somente enquanto durar a execução da aplicação. Estes recursos podem ser depois realocados para aplicações diferentes. Além disso, para permitir a execução de um número amplo de aplicações de diferentes tipos, é essencial que a configuração da infraestrutura, inclusive a instalação de qualquer componente de *software* específico da aplicação, possa ser realizada de forma simples e ágil. Tal premissa deve continuar válida até mesmo considerando-se que a escala desejada esteja na ordem de milhões de nós de processamento. Em outras palavras, o usuário deve ser capaz de facilmente e rapidamente personalizar a infraestrutura de processamento inteira de acordo com as suas necessidades.

Em resumo, para prover suporte adequado a um escopo amplo de aplicações HTC, nós contemplamos que um *JiT DC* de alta vazão precisa satisfazer os seguintes requisitos:

1. *escalabidade extremamente alta*: deve poder controlar até milhões de nós de proces-

samento da mesma forma que controla algumas dezenas ou centenas deles;

2. *instanciação sob demanda*: precisa oferecer mecanismos para descoberta, montagem e coordenação dos recursos solicitados, sob demanda e por uma quantidade específica de tempo; e,

3. *configuração eficiente*: a configuração dos dispositivos de processamento deve ser levada a termo com rapidez e com um mínimo de esforço, não exigindo nenhuma intervenção individual ou especializada.

Infelizmente, as tecnologias atuais que poderiam ser usados neste caso, tanto as baseadas em recursos oportunistas, como grades de *desktops* e computação voluntária, quanto as baseadas em recursos dedicados, como IaaS, possuem limitações fundamentais que têm impactos ou na sua escala ou no seu alcance.

Embora as grades de *desktops* forneçam os mecanismos necessários para a instanciação sob demanda, suas principais limitações são a configuração lenta e a escalabilidade relativamente baixa. A personalização do ambiente de processamento é demorada, uma vez que cada recurso precisa ser configurado individualmente, sempre que uma mudança é necessária. Uma vez que os recursos são distribuídos por diferentes domínios administrativos, cada um impondo suas políticas de segurança próprias, é mais difícil fazer com que um grande número de provedores de recursos cheguem a um consenso sobre um conjunto de políticas compatíveis. Além disso, em grades de *desktops* um comportamento de reciprocidade é esperado e há a necessidade de controles adicionais sobre a forma como os recursos da rede são compartilhados, de forma a inibir o surgimento de caronistas (*free riders*) [Andrade et al. 2007].

Os sistemas para computação voluntária [Anderson et al. 2002; Anderson 2004] provaram que é possível construir plataformas computacionais com milhões de nós para suportar a execução de aplicações HTC. Estes sistemas, entretanto, não possuem a flexibilidade das infraestruturas de grades de *desktops* [Litzkow, Livny e Mutka 1988; Cirne et al. 2006; Oliveira, Lopes e Silva 2002; Andrade et al. 2007; Thain, Tannenbaum e Livny 2006], sendo uma solução válida somente para um subconjunto muito pequeno de aplicações que podem se beneficiar da vazão extremamente alta que eles podem entregar. A abordagem de computação voluntária tem sido bem sucedida apenas nos casos onde a aplicação possui

um apelo que motive os usuários a participarem dos projetos e doarem recursos computacionais para os projetos. Os casos de sucesso mais relevantes envolvem a busca pela cura de doenças [Stanford 2011] e busca por vida extraterrestre [Anderson et al. 2002].

Mais recentemente, IaaS também se apresentou como uma tecnologia apta para a instanciação sob demanda de infraestruturas computacionais [Wang et al. 2010]. Algumas companhias já oferecem a possibilidade de configurar sistemas compostos por um grande número de máquinas virtuais, fornecendo uma interface similar a grades computacionais [Amazon 2010]. Isto facilita o esforço de montar um grande conjunto de servidores, que podem ser substituídos por máquinas virtuais hospedadas em centros de dados de fornecedores de IaaS. Embora sejam, em tese, virtualmente inesgotáveis, estas infraestruturas estão limitadas tanto pela capacidade física dos provedores atuais quanto pelos modelos de negócios vigentes, que restringem a alocação de uma quantidade muito alta de nós de processamento, conforme foi discutido no Capítulo 2. Embora muito flexíveis e simples de configurar, ativar computação de vazão extremamente alta em IaaS não é tão automático considerando-se as implementações disponíveis.

No caso especial dos requisitos específicos para a construção de *JiT DCs* de alta vazão, a Tabela 5.1 mostra como as tecnologias atualmente disponíveis endereçam os requisitos identificados apenas parcialmente. Como pode ser observado, todos os requisitos são atendidos por pelo menos uma das soluções disponíveis, mas nenhuma das tecnologia citadas é capaz de atender, adequada e simultaneamente, a todos eles.

Tabela 5.1: Tecnologias Disponíveis x Requisitos

Requisitos	**Tecnologias Disponíveis**		
	Computação Voluntária	***Desktop Grids***	***IaaS***
Escalabidade Extremamente Alta	×		
Configuração Eficiente			×
Instanciação sob Demanda		×	×

5.2 Infraestrutura Computacional Distribuída Sob Demanda (OddCI)

Nesta seção nós apresentaremos uma nova arquitetura para construir *JiT DCs* dinâmicos[3] baseados em recursos computacionais de alta granularidade, alta volatilidade e alta dispersão que é, ao mesmo tempo flexível e altamente escalável, sendo aplicável para a execução eficiente de aplicações BoT de larga escala e curta duração. Com esta abordagem, um cliente poderá alocar, sob demanda, um conjunto com um grande número de unidades de processamento, chamada de instância DCI, que executará sua aplicação BoT de forma tão eficiente quanto possível. Após completar a execução, o cliente liberará a instância DCI que foi criada. Por causa desta singularidade, a arquitetura é chamada de *Infraestrutura Computacional Distribuída Sob Demanda* (ou OddCI, do inglês *On-Demand Distributed Computing Infrastructure*).

A arquitetura OddCI é formada por um *Provider*, um *Backend*, uma ou mais redes de *broadcast*, cada uma contendo um canal de *broadcast* e um *Controller*, e *Processing Node Agents* (PNA). Estes últimos são programas a serem enviados e executados em cada um dos recursos computacionais acessíveis pelo *Controller* através da sua rede de *broadcast* correspondente. Além disso, é assumido que os recursos computacionais também possuem um canal bidirecional, chamado de *canal direto*, o qual os conecta tanto com o *Backend* quanto com o seu respectivo *Controller* (Fig. 5.1).

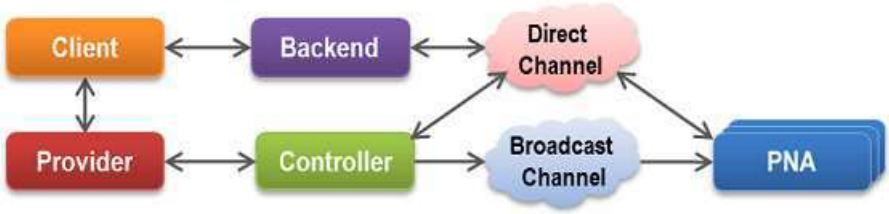


Figura 5.1: Visão Geral da Arquitetura OddCI

A seguir, é feita uma breve descrição de cada um dos componentes previstos na arquitetura OddCI:

[3] A partir deste ponto do documento, usaremos o termo *JiT DC dinâmicos* para nos referirmos a JiT DCs de alta vazão baseados em recursos de alta granularidade, alta volatilidade e alta dispersão no contexto de redes de *broadcast*.

- O *Provider* (provedor) é responsável por criar, gerenciar e destruir as instâncias OddCI de acordo com as solicitações dos clientes e também pela autenticação do cliente e pela verificação das suas credenciais para usar os recursos que estão sendo requisitados;

- O *Controller* (controlador) é encarregado de configurar a infraestrutura, conforme instruído pelo *Provider*, através da formatação e envio, via canal de *broadcast*, de mensagens de controle e imagens de *software* (executáveis) para os dispositivos, necessárias para construir e manter as instâncias OddCI;

- O *Backend* (retaguarda) é responsável pelo gerenciamento das atividades específicas de cada aplicação sendo executada. Estas atividades podem incluir a distribuição (escalonamento) de tarefas, o provisionamento de dados de entrada bem como a recepção e, eventualmente, o pós-processamento dos resultados gerados pela aplicação paralela;

- *Processing Node Agents* (PNA) (agentes processadores) são responsáveis pelo gerenciamento da execução da aplicação do cliente no dispositivo computacional e o envio de sondas periódicas (*heartbeat messages*) para sinalizar o seu estado;

- O *Direct Channel* (canal direto), por sua vez, é uma rede de comunicação bidirecional que permite a comunicação entre todos os componentes da arquitetura, tal como a Internet; e,

- O *Broadcast Channel* (canal de *broadcast*) é um canal unidirecional para envio de dados do *Controller* para os dispositivos. Pode ser, por exemplo, um canal de TV Digital ou uma estação rádio base (ERB) de uma rede celular.

Os dispositivos que executarão o PNA são descobertos e inicializados através de uma *wakeup message* (WM) transmitida pelo *Controller*. Esta mensagem de controle contém, dentre outras coisas, o executável do PNA e a imagem da aplicação do cliente. Um PNA está estruturado como ilustrado na Fig. 5.2.

O *Monitor* interage, de forma passiva, com o *Controller* através do canal de *broadcast*, processando as mensagens de controle recebidas, carregando novas imagens de aplicações em um DVE (do inglês, *Dynamic Virtual Environment*) [Keahey, Doering e Foster 2004] e gerenciando a execução da imagem carregada. O *Monitor* se comunica com o *Controller*, de forma ativa, através do canal direto para relatar seu estado atual. O DVE habilita um

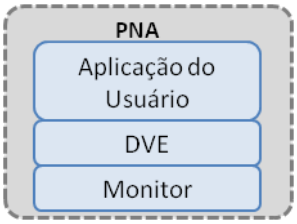


Figura 5.2: Estrutura Interna de um PNA

ambiente seguro e adequado para execução da aplicação do usuário OddCI, no intuito de salvaguardar os interesses do proprietário do dispositivo, do cliente e do operador da rede de *broadcast*. Finalmente, a *Aplicação do Usuário* é a imagem da aplicação que é carregada no PNA e que realiza o processamento específico desejado pelo cliente.

5.2.1 Funcionamento OddCI

O funcionamento básico de um sistema OddCI (criação e operação) pode ser observado através dos fluxos de troca de mensagens possíveis entre os seus componentes, conforme ilustrado na Fig. 5.3.

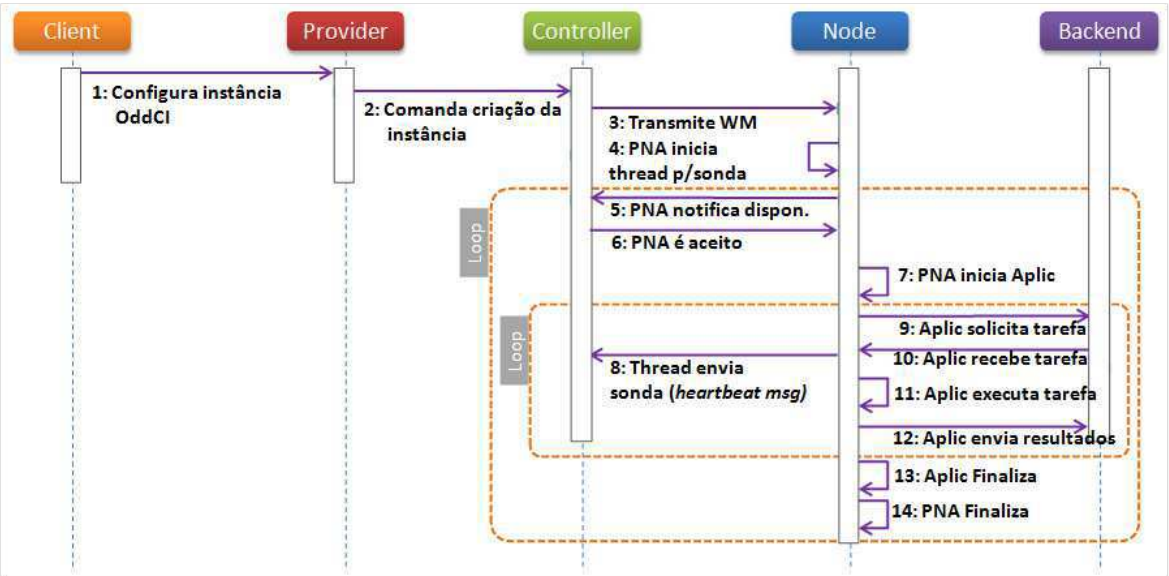


Figura 5.3: Fluxo de Operação OddCI

Um *Client* OddCI interage com o sistema usando uma interface implementada pelo *Provider*. A interface pode ser usada para instruir o *Provider* para criar instâncias OddCI personalizadas para as necessidades do usuário.

Inicialmente, o *Client* submete ao *Provider* um pedido para a criação de uma instância OddCI, indicando os requisitos para os dispositivos e fornecendo uma imagem de aplicação específica, incluindo programas, dados comuns e o tamanho desejado da instância. A solicitação do *Client* também fornece as credenciais do usuário, de forma que a autenticação e os procedimentos de segurança e controle de acesso possam ser executados.

Ao receber um pedido para criar uma nova instância OddCI, o *Provider* autentica o *Client*, valida a imagem da aplicação e, baseado no histórico e em estimativas dos recursos disponíveis no momento, decide se o pedido pode ser atendido ou não. Se ele prevê que existam recursos suficientes, ele encaminha o pedido para o *Controller* apropriado para alocar recursos e criar a instância OddCI.

Depois de validar o *Provider* e o pedido da instância, o *Controller* formata uma *wakeup message* adequada, a qual contém todas as informações relevantes, extraídas do pedido da instância, referentes à aplicação do cliente, bem como um PNA configurado para suportar a nova instância OddCI a ser criada. Esta mensagem de controle é enviada através do canal de *broadcast*. Este processo é chamado de *wakeup process*, ou "despertar", de uma instância OddCI.

Um dispositivo é configurado para somente aceitar mensagens transmitidas pelo seu respectivo *Controller*[4]. Se um PNA já está em execução em um recurso computacional, então qualquer nova WM recebida é descartada. Caso contrário, o recurso computacional carrega o PNA e inicia a sua execução.

Então, o PNA avalia a sua própria conformidade com os requisitos presentes na mensagem e, se houver compatibilidade, ele usa o canal direto para sinalizar para o *Controller* a sua disponibilidade para ser integrado à instância OddCI. O *Controller* irá responder aceitando ou liberando o PNA. Se aceito, o PNA cria um DVE para a carga e execução da aplicação do cliente presente na WM recebida. Enquanto a aplicação está rodando, o PNA periodicamente envia sondas (*heartbeat messages*) para o seu *Controller* através do canal direto, sinalizando que está ativo. Tais mensagens contêm o estado do PNA e a identificação da instância OddCI à qual o mesmo pertence atualmente. O intervalo de tempo entre o envio de duas *heartbeat messages*, chamado *heartbeat interval*, é determinado pelo *Controller* na própria WM. Através da consolidação das sondas recebidas de todos os PNAs pertencentes

[4]Isto pode ser obtido através de um mecanismo baseado em assinatura digital de mensagens.

a uma determinada instância OddCI, o *Controller* pode monitorar o seu tamanho e enviar novas WMs para adicionar novos dispositivos à instância sempre que necessário.

Deste ponto em diante, a aplicação pode se comunicar com o *Backend* diretamente através do canal direto para buscar novas tarefas [5] e transmitir os resultados processados. Quando não há mais tarefas disponíveis, a aplicação finaliza a sua execução, e assim também faz o PNA.

O *Controller* também pode transmitir mensagens de controle do tipo *reset message* (RM) para destruir uma instância OddCI em particular. Após receber uma RM, um PNA que integra a instância específica, interrompe a execução da aplicação, destrói o DVE e finaliza a sua execução. Além disso, o *Controller* também pode descartar PNAs individualmente através do canal direto, durante o tratamento de *heartbeat messages*, com o objetivo de ajustar uma instância OddCI cujo tamanho esteja acima do desejado. Da mesma forma, o *Controller* pode necessitar retransmitir WMs periodicamente para recompor instâncias OddCI que perderam alguns dos seus PNAs, uma vez que os recursos computacionais usados não são, necessariamente, assumidos como dedicados, e podem ser desligados sem aviso prévio, de acordo com a vontade dos seus proprietários.

5.3 Aspectos de Segurança

A segurança é uma questão importante a ser considerada na concepção e implementação de um sistema OddCI. Cada ator de um sistema OddCI possui as suas próprias expectativas e interesses em matéria de segurança. Os clientes (*Clients*) esperam que a sua aplicação e os dados associados estejam protegidos durante todo o ciclo de vida de uma instância OddCI. Além disso, eles precisam se proteger contra resultados espúrios fornecidos por sabotadores ou recursos computacionais defeituosos. O fornecedor do serviço OddCI (*Provider*) precisa autenticar os clientes, suas aplicações, bem como os controladores (*Controllers*) que usa. Os controladores devem evitar perturbações no seu funcionamento causado por sondas indevidas oriundas de PNAs executando em dispositivos computacionais comprometidos ou com mal funcionamento. Finalmente, os proprietários dos equipamentos que executam os PNAs e as

[5] Nós usamos o termo tarefas para nos referirmos a quaisquer dados adicionais que a aplicação demande do *Backend*.

aplicações precisam de garantias de que a execução destas aplicações não vai interferir com o funcionamento de seus dispositivos (exibição de forma adequada da programação de TV, no caso de receptores de TV digital, por exemplo).

5.3.1 Requisitos de Segurança

Os requisitos de segurança que precisam ser atendidos em nosso contexto podem ser consolidados a partir da observação da dinâmica de interações entre os componentes de um sistema OddCI. A Fig. 5.4 traz as interações básicas entre estes componentes.

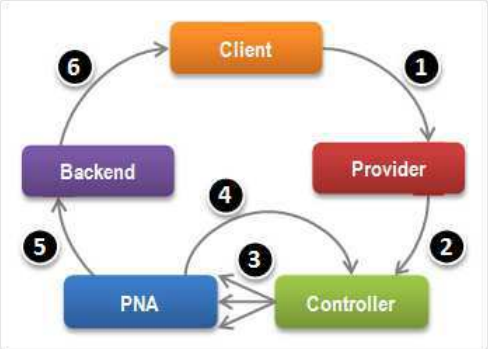


Figura 5.4: Interações Básicas entre os Participantes de um Sistema OddCI

O fluxo (**1**) requer a autenticação mútua entre o *Client* e o *Provider*, e a confidencialidade na comunicação, entre eles como forma de proteger a imagem (código a ser executado) e os dados enviados para o *Provider*. No fluxo (**2**), autenticação mútua também é necessária entre o *Controller* e o *Provider*, bem como a confidencialidade na troca de mensagens de controle. No fluxo (**3**), o PNA precisa receber mensagens de forma confidencial, bem como autenticar a origem das mensagens de controle recebidas, visando garantir que elas são realmente oriundas do *Controller* apropriado. Nos fluxos (**4**) e (**5**), o PNA e a aplicação precisam de autenticação e confidencialidade para estabelecer comunicações seguras com o *Controller* e o *Backend*, respectivamente. Finalmente, o fluxo (**6**) envolve uma comunicação particular e controlada entre o *Client* e a sua estrutura de retaguarda (*Backend*). Esta fora do escopo deste trabalho discutir como a mesma pode ser realizada, entretanto, pelas suas características, o mesmo tratamento aplicado nos fluxos (**1**) e (**2**) também pode ser utilizado.

Nos fluxos de comunicação "um-para-um" (1, 2, 4 e 5), autenticação e confidencialidade podem ser obtidas com facilidade se as partes envolvidas puderem ser devidamente identi-

ficadas. Este é o caso para os fluxos 1 e 2 mas não para os fluxos 3 e 4. Como o PNA é um componente volátil, não conhecido previamente, a sua autenticação precisa ser tratada de forma especial[6]. Além disso, o canal de *broadcast* estabelece uma comunicação de "um-para-muitos" entre o *Controller* e os PNAs, a qual requer mecanismos de autenticação e confidencialidade distintos dos usados nos fluxos "um-para-um".

A confidencialidade da imagem da aplicação precisa ser garantida até a sua efetiva execução, sendo transversal para os fluxos (**1**), (**2**) e (**3**). Confidencialidade transversal, neste caso, significa que a mensagem seja enviada, sequencialmente, da parte 1 para a parte N, mas que só possa ser aberta pelo destino final (Princípio da Não Interferência Intransitiva [Schellhorn et al. 2002]). Por exemplo, somente a aplicação cliente instanciada pelo PNA deve ser capaz de decriptografar os dados da aplicação enviados pelo *Client* e retransmitidos pelo *Provider* e pelo *Controller*.

Adicionalmente, o *Backend* precisa validar a integridade dos resultados recebidos para se proteger de falhas Bizantinas [Sens 2010] ou tentativas de sabotagem [Sarmenta 2001], as quais podem exigir controles específicos que consideram a semântica e a sintaxe adotada em cada aplicação.

A Tabela 5.2 traz um sumário dos objetivos de segurança extraídos dos requisitos levantados.

[6]O uso de mecanismos de autenticação especiais (usando conceitos como chaves embutidas (*embedded keys*) [Boesgaard e Zenner 2007] e ofuscamento de programas [D'Anna et al. 2003], por exemplo) inseridos dentro do código do PNA e da aplicação é uma alternativa de associar uma identidade para estes processos que executam nas partes não controladas do sistema, tornando-as passíveis de serem autenticadas pelos processos de retaguarda equivalentes no *Controller* e no *Backend*. O uso das técnicas de chaves embutidas e de ofuscamento, além de aplicável, ganha uma vantagem adicional no contexto OddCI no qual as instâncias são formadas dinamicamente. Como o código do PNA e da aplicação fornecida pelo cliente são enviados em cada WM, as chaves embutidas e a técnica de ofuscamento podem ser alteradas frequentemente para ficarem obsoletas com rapidez. Isto reduz o tempo de exposição de tais mecanismos e diminui a eficácia de ataques destinados a obter tais chaves e interferir na comunicação entre o *Controller* e o PNA e entre a aplicação e a sua retaguarda.

[7]Bloqueante, neste caso, significa que a parte que receberá uma mensagem fica bloqueada, esperando a mensagem chegar.

Tabela 5.2: Objetivos de Segurança

ID	Objetivos de Segurança
O1	Autenticação mútua de partes previamente identificadas nos fluxos (**1**) e (**2**)
O2	Autenticação unilateral de partes previamente identificadas no fluxo (**3**)
O3	Autenticação unilateral de partes voláteis e não identificadas nos fluxos (**4**) e (**5**)
O4	Comunicação bloqueante[7] segura para os fluxos (**1**), (**2**), (**4**) e (**5**)
O5	Comunicação não bloqueante segura para o fluxo (**3**)
O6	Comunicação transversal segura para os fluxos (**1**), (**2**) e (**3**)
O7	Controle semântico fim-a-fim no fluxo (**5**)
O8	Confidencialidade e integridade em todos os fluxos

5.3.2 Modelo de Segurança

No modelo de segurança descrito nesta seção, nós propomos um conjunto de primitivas e um protocolo de uso que permitem atender os requisitos de segurança envolvidos no fluxo operacional de um sistema OddCI[8].

Primitivas

As primitivas de segurança necessárias para o atendimento dos objetivos de segurança identificados na seção anterior estão relacionadas na Tabela 5.3. É assumido que tais primitivas são plenamente suportadas pelos recursos computacionais de um Sistema OddCI[9].

Protocolos de Segurança

O modelo de segurança que estamos propondo é baseado em camadas de "envelopes" criptográficos e técnicas de controle fim-a-fim que permitem ativar autenticação, confidencialidade e também proteção contra falhas e sabotagens.

Incialmente, o *Client* U solicita ao *Provider* P a criação de uma instância OddCI I. Se a operação é bem sucedida, o *Provider* retorna um identificador único da instância criada

[8]Não está contemplada aqui a abordagem de ameaças físicas de nenhuma natureza nem ameaças em nível de corrupção de *hardware* ou *software* básico, reuso de memória ou acesso direto a registradores internos.

[9]Observe que essas primitivas não precisam ser implementadas como funções atômicas suportadas pelos recursos computacionais.

Tabela 5.3: Primitivas Básicas de Segurança

Primitiva	**Descrição**
Hash(m)	Calcula um *hash* não inversível para a mensagem *m*
Crypt(m, k)	Cifra a mensagem *m* usando a chave *k*
DeCrypt(m, k)	Decifra a mensagem *m* usando a chave *k*
KeyGen(id_1, id_2)	Gera uma chave para uso em sessão de comunicação entre as identidades id_1 e id_2
SecureChannel(d)	Estabelece um canal de comunicação seguro com o destino *d*. O canal poderá ser usado para envio de mensagens subsequentes. O estabelecimento do canal seguro pré-supõe a autenticação mútua dos parceiros envolvidos
SecureSend(S, m)	Envia uma mensagem *m* usando o canal seguro *S*
SecureReceive(S)	Recebe uma mensagem *m* através do canal seguro *S*
PublicKey(id)	Retorna a chave pública associada à identidade *id*
Sign(m, k)	Assina a mensagem *m* usando a chave privada *k*
Verify(m, id)	Verifica a autenticidade e integridade da mensagem *m* assinada pelo autor *id* e retorna VERDADEIRO, caso a checagem seja bem sucedida, ou FALSO, caso contrário
Auth(id)	Verifica a autenticidade da identidade *id* mediante algum protocolo baseado na troca síncrona de certificados de autenticação ou equivalente
FormatImage(e, d)	Cria uma imagem usando o executável *e* e os dados *d*
CreateInstance(S, I)	Solicita a criação de uma instância OddCI *I* através do canal seguro *S*. Assume-se que o canal seguro é estabelecido com um elemento do tipo *Provider*
Broadcast(B, m)	Envia a mensagem *m* pelo canal de *broadcast B*
ProcessID(p, id)	Vincula um processo *p* à identidade *id* através de algum mecanismo que permita a inserção de *tokens* embutidos no código binário da aplicação

(OddCI_ID). O *Client* arbitra uma chave (BackendKey) a ser usada na comunicação com o *Backend* para acesso às tarefas e resultados (*BackendKey*) e embute esta chave no executável da sua aplicação que rodará nos PNAs da instância I. O *Client* também acrescenta nos dados da aplicação informações sobre os endereços dos servidores que compõem a infraestrutura do *Backend*. O *Backend* usará a mesma chave para autenticar os dispositivos computacionais que em breve se conectarão para estabelecer um canal seguro de comunicação para recepção de novas tarefas e envio de resultados. Em seguida, um envelope é criado pelo *Client* para conter os dados da sua aplicação, o qual é enviado para o *Provider* P. Salienta-se que o estabelecimento do canal seguro assume a prévia autenticação mútua das partes envolvidas, como apresentado na Tabela 5.2. A sequência de primitivas abaixo representa o que foi discutido.

```
sc_provider = SecureChannel(P)
OddCI_ID  = CreateInstance(sc_provider, I)
ExecutableKey = ProcessID(Executable, BackendKey)
AppImage = FormatImage(ExecutableKey,
                       Crypt(data, BackendKey)
                      )
SecureSend(sc_provider, AppImage)
```

Do lado do *Provider* P, a mensagem do *Client* U é recebida de forma confidencial como segue:

```
sc_client = SecureChannel(U)
AppImage = SecureReceive(sc_client)
```

O passo seguinte para o *Provider* P é repassar para o *Controller* C uma mensagem de controle contendo a imagem da aplicação e instruções sobre o tipo de instância a ser criada.

```
ControlMessage = Format(AppImage, params,
                        OddCI_ID)
sc_controller = SecureChannel(C)
SecureSend(sc_controller, ControlMessage)
```

O *Controller* C recupera a mensagem de controle (fluxo 2), gera uma chave randômica exclusiva (InstanceKey) para a instância OddCI_ID e a embute no código do PNA. Na prática essas informações servirão de credenciais para autenticar cada PNA, de maneira que o controlador apenas aceitará como participante da instância o PNA que apresentar a InstanceKey

correta como credencial. Em seguida, o controlador *Controller* C formata, cifra e depois assina a mensagem de controle recebida do *Provider* P e a propaga através do canal de *broadcast* para todos os dispositivos conectados.

```
sc_provider = SecureChannel(P)
ControlMessage = SecureReceive(sc_provider)
InstanceKey = Random(OddCI_ID)
PNAwKey = ProcessID(PNA, InstanceKey)
ControlMessage = Format(ControlMessage, PNAwKey)
M = Crypt(Sign(ControlMessage, Kprivc)
SignControlMessage = Sign(M, Kprivc)
Broadcast(BroadcastChannel, SignControlMessage)
```

Todos os dispositivos conectados ao canal de *broadcast* recebem a mensagem que contém a aplicação assinada. Conforme o fluxo operacional OddCI descrito anteriormente, o dispositivo fará a validação da mensagem usando a chave pública do *Controller*, a qual está autenticada por uma autoridade certificadora previamente estabelecida. Uma vez que a mensagem é validada pelo dispositivo, o PNA é então carregado, e faz a comunicação com o *Controller* usando o identificador InstanceKey, o qual foi previamente embutido no seu código, como chave para garantir a autenticação e o sigilo no fluxo 4.

O passo seguinte do PNA, caso seja aceito pelo *Controller* para participar da instância I, é iniciar a aplicação propriamente dita, a qual está de posse da chave BackendKey, e pode finalmente abrir o primeiro envelope criado pelo *Client* para recuperar os dados da aplicação. Esta mesma chave é usada como identificador para estabelecer um canal seguro com o *Backend* através do fluxo 5. Para minimizar o fato de que um PNA com uma chave embutida que é enviado através da rede de *broadcast* pode ser capturado por qualquer pessoa e, posteriormente, usado para emitir mensagens de controle espúrias, optou-se pela utilização de uma chave transitória e individualizada para cada instância. Assim, mesmo que um atacante possa quebrar o ofuscamento e recuperar uma InstanceKey ainda durante o tempo de vida da instância associada, possíveis ataques, como o envio de sondas falsos para o *Controller*, são limitadas no tempo e na abrangência.

As chaves embutidas na aplicação (BackendKey) e no PNA (InstanceKey), criadas de forma exclusiva e independente pelo *Client* para cada aplicação e pelo *Controller* para cada instância OddCI, representam uma adaptação do conceito de "trusted process" proposto por

Bell/LaPadula [Bell e LaPadula 1976; Lunt, Neumann e al. 1998], e permitem a validação dos elementos voláteis do sistema. Embora estas chaves específicas tenham um ciclo de vida curto e estejam embutidas nos respectivos executáveis, elas ainda representam uma fragilidade. Estas são as únicas chaves potencialmente acessíveis a partir de nós remotos que poderiam ser obtidas via engenharia reversa dos executáveis ou varredura de memória em dispositivos computacionais comprometidos. Entretanto, as técnicas propostas por Boesgaard *et al.* [Boesgaard e Zenner 2007] podem ser utilizadas para tornar muito mais improvável que ataques deste tipo sejam bem sucedidos.

Além destes mecanismos, o tratamento de falhas Bizantinas [Sens 2010] e técnicas de controle de sabotagem [Sarmenta 2001] são aplicados nos fluxos 4 e 5, encapsuladas em controles semânticos fim-a-fim. Usando controles deste tipo, o *Backend* pode enviar tarefas especiais e conferir os resultados recebidos para validar cada PNA ou criar certa quantidade de réplicas das tarefas e enviá-las para serem processadas por mais de um PNA. Somente quando um número de resultados convergirem (por exemplo, a maioria), a tarefa é considerada completa. A quantidade de réplicas pode ser manipulada para se adaptar a contextos com maior ou menor grau de suscetibilidade a ataques de adversários. A estratégia de controle fim-a-fim adotada, independentemente da sua forma de implementação, deverá ficar localizada na distribuição de tarefas e recolhimento de resultados de cada *Backend* específico.

5.4 Aspectos de Implementação

5.4.1 Disponibilidade Coletiva

No contexto OddCI considerado, os recursos alocados para processar aplicações paralelas podem ser voláteis, assim, ao longo do tempo, o conjunto de recursos alocados em qualquer instância OddCI pode reduzir de tamanho. Portanto, é necessário reparar a perda esperada de recursos através de uma estratégia de antecipação ou de compensação, que chamamos de algoritmos compensatórios.

A utilização de métodos de predição para suportar mecanismos que assegurem a disponibilidade coletiva (*collective availability* [Andrzejak, Kondo e Anderson 2008]) de uma coleção volátil de recursos tem sido estudada por Andrzejak *et al.* O estudo mostra que

usando métodos adequados de previsão, é possível garantir que um subconjunto qualquer de nós de tamanho não menor do que ω em um conjunto volátil Ω esteja disponível durante um período de tempo de tamanho $\triangle T$, com uma sobrecarga (*overhead*) de controle razoável.

A taxa de sucesso (*success rate*) obtida quando se tenta manter pelo menos ω dispositivos disponíveis em um dado período de tempo é dependente do tempo médio de disponibilidade dos dispositivos do conjunto volátil Ω (*historical turnover rate*) e do valor de ω, mas pode ser equilibrada através de um nível adequado de redundância, R, alocando $\omega + R$ recursos. Os resultados apresentados por Andrzejak *et al.* indicam que a solução mais prática para controlar a disponibilidade coletiva é uma combinação de uma abordagem de previsão simplificada com o ranqueamento dos dispositivos de acordo com o seu comportamento histórico de disponibilidade. Com base nisso, uma sequência de *bits* pode ser usada para representar a disponibilidade histórica de cada dispositivo em instantes de tempo específicos e um modelo de predição processa as sequências de *bits* dos dispositivos, gerando um *ranking* de regularidade que pode ser usado para instruir o processo de seleção de recursos, de forma que sejam atendidos requisitos de disponibilidade específicos.

Em nossa abordagem, uma variação escalável desse método é obtida através do registro das informações históricas de disponibilidade pelo próprio PNA. A alocação inicial de recursos para criar uma instância com $\omega + R$ dispositivos é realizado em um único passo pelo *Controller* que envia para os recursos as informações necessárias, incluindo o alvo de disponibilidade desejado, através de uma WM. Este processo pode ser repetido várias vezes durante o ciclo de vida da instância para recuperar eventuais perdas de dispositivos e manter a instância no tamanho requisitado. O valor R é dinamicamente definido em cada *wakeup process*, considerando a taxa de perda de recursos observada e o tempo necessário para transmitir a WM.

No entanto, uma WM pode ativar uma instância com um número de recursos que é muito maior ou muito menor do que o necessário, dependendo da disponibilidade instantânea de recursos. Qualquer quantidade excedente de PNAs que respondam à WM será descartado pelo *Controller*. Da mesma forma, a detecção de que uma quantidade menor de PNAs do que a necessária respondeu a WM irá desencadear novas tentativas de alocação de recursos através do envio de novas WMs.

5.4.2 Estratégias de Escalonamento e Provisionamento

A eficiência do *Provider* está relacionada com a forma como ele escalona e monitora as instâncias OddCI delegadas para os *Controllers* do sistema OddCI. Após receber uma solicitação de um *Client*, o *Provider* deve selecionar o subconjunto de *Controllers* capazes de lidar com os requisitos solicitados, e também definir quais deles devem ser escolhidos para atender a instância OddCI, considerando tanto o cumprimento do SLA estabelecido, bem como garantir um melhor resultado operacional, ou seja, reduzindo a redundância necessária a um valor mais próximo do mínimo exigido.

Quando um *Client* submete um pedido para criação de uma instância OddCI, ele define os requisitos desejados para os recursos (tipo, quantidade, etc) em uma OIR (*OddCI Instantiation Request*).

No contexto OddCI, a estratégia usada pelo *Provider* para distribuir as OIR pelo conjunto de *Controllers* é chamada *estratégia de escalonamento*. Esta estratégia pode ser implementada pragmaticamente através do uso de uma função de custo que é capaz de implementar uma avaliação dos critérios desejados sobre o conjunto de *Controllers* disponíveis.

Seja $f(O, C_i)$ uma função que retorna verdadeiro ou falso, dependendo se o *Controller* C_i pode ou não pode atender a OIR O, e $c(O, C_I)$ seja a função de custo para a criação de O em C_i. O *Controller* C_i é escolhido se:

$$f(O, C_i) \wedge \ \nexists\, C_j \mid f(O, C_j) \ \wedge \ c(O, C_j) \ < \ c(O, C_i).$$

Dependendo da estratégia para a seleção do *Controller*, a função c pode ser definida de modo a refletir os critérios desejados. Por exemplo, o custo estimado pode refletir tanto um critério mais direto, como o preço a ser pago pelo *Provider* para cada *slot* de processamento usado em uma rede de *broadcast* específica, e também pode considerar aspectos mais complexos, tais como o risco do *Provider* de incorrer no pagamento de eventuais sanções por não cumprir com a OIR ou o custo envolvido pela necessidade do *Provider* ter que usar um excedente de recursos para manter o tamanho da instância nos níveis adequados.

Por sua vez, o *Controller* deve tentar manter o nível real de paralelismo (P_R), ou tamanho da instância, durante o seu ciclo de vida tão perto quanto possível do nível de paralelismo solicitado (P_S) para evitar violações do SLA. O tamanho da instância é definido pela quan-

tidade de dispositivos ativos que ela contém em um dado momento. Baseando-se tanto em informações instantâneas enviadas pelos PNAs quanto em dados históricos, o *Controller* precisa disparar as mensagens de controle necessárias para coordenar esse equilíbrio. Nós chamamos este procedimento de *estratégia de provisionamento*.

Por suas características únicas e considerando um cenário *best-effort*, o custo de migração de recursos computacionais em um sistema OddCI é o mesmo, independentemente da quantidade de recursos computacionais que foram perdidos. Isto ocorre porque o esforço envolvido em um *wakeup process* é praticamente o mesmo, seja a WM destinada a alocar um ou um milhão de dispositivos. A sua duração depende unicamente do tamanho da imagem da aplicação e da largura de banda do canal de *broadcast*. No entanto, essa característica traz consigo uma sobrecarga de coordenação potencial, porque qualquer excedente de dispositivos ativado pela WM deve ser eliminado pelo *Controller*, e isto é realizado trocando mensagens através do canal direto. Esta operação consome recursos dos dispositivos, do canal direto, e do *Controller*. Tal sobrecarga deve ser minimizada.

Para o bom funcionamento das estratégias de provisionamento, é essencial que o *Controller* tenha uma boa aproximação da população de recursos à disposição (Ω), da redundância necessária (R), e do número total de recursos que serão potencialmente afetados pelo *wakeup process*. Uma vez estimado o valor de $|\Omega|$ e definido o valor de P_S para incluir R, onde $P_S + R < |\Omega|$, é importante tomar cuidado para que somente $P_S + R$ recursos respondam a uma WM, apesar de todos os recursos conectados ao canal de *broadcast* de um dado *Controller* receberem a WM transmitida pelo canal. Este problema torna-se mais crítico nos casos em que $P_S + R << |\Omega|$.

Uma estratégia simples para acionar apenas um subconjunto de tamanho aproximadamente igual a $P_S + R$ numa população alvo de tamanho $|\Omega|$ é enviar, com a WM, um fator probabilístico p de tal forma que cada recurso que recebe a WM a descarta com probabilidade $1 - p$. O valor de p pode ser inicialmente determinado pela razão entre P_S e $|\Omega|$ e ajustado em rodadas sucessivas, considerando também o número de recursos que respondem a WM, o qual será utilizado para melhorar a estimativa de $|\Omega|$.

Com o uso de ranqueamento, o critério de elegibilidade do PNA primeiro verifica o *ranking* do dispositivo e depois aplica o fator probabilístico indicado em p, o qual deve ter sido calculado considerando uma estimativa da quantidade de dispositivos disponíveis que

atendem ao alvo de ranqueamento desejado. Eventualmente, o *Controller* pode precisar diminuir o *ranking*-alvo para ajustá-lo à condição atual de ranqueamento dos dispositivos disponíveis e conseguir obter a quantidade necessária de dispositivos para repor o tamanho da instância.

Após a criação da instância, o *Provider* mantém contato com os *Controllers* a fim de monitorar os requisitos solicitados. Se necessário e possível, o *Provider* pode redistribuir instâncias OddCI entre *Controllers* para refletir um novo estado do sistema causado pela criação e desmonte de outras instâncias OddCI, a perda de dispositivos das várias redes de *broadcast* etc. Isto pode envolver a avaliação de escalonamento alternativo para a instância, com a possível seleção de outros *Controllers*. Portanto, a estratégia de escalonamento deve ser cuidadosamente projetada para otimizar o uso dos recursos disponíveis, levando em consideração o contexto em que o OddCI está sendo implantado de forma a minimizar os custos do *Provider* e maximizar a sua eficiência.

5.5 Avaliando o Desempenho do Sistema

O objetivo principal da nossa avaliação foi investigar o potencial de uso de recursos tercerizados em *JiT Clouds* no cenário mais desafiador, caracterizado por alta granularidade, alta volatilidade e alta dispersão através do uso da arquitetura OddCI para a sua descoberta, alocação e coordenação.

Nós descrevemos nas próximas subseções como esta avaliação foi projetada e realizada através de simulação.

5.5.1 Modelo de Simulação

Nesta subseção é feita uma descrição mais formal do modelo de operação de sistemas OddCI que foi utilizado na nossa simulação.

Consideramos uma rede de *broadcast* que pode acessar um conjunto $\mathbb{D}$ de dispositivos. Seja $A\left(d,t\right)$ uma função boleana no tempo que indica se um dispositivo $d \in \mathbb{D}$ está ativo no momento t. O conjunto de dispositivos *ativos* no momento t, $\mathbb{D}^a\left(t\right)$, é dado por $\mathbb{D}^a\left(t\right) = \{\ d \mid d\ \in \mathbb{D} \bigwedge A\left(d,t\right) = true\}$ e o conjunto de dispositivos *inativos* no momento t, $\mathbb{D}^i\left(t\right)$, é dado por $\mathbb{D}^i\left(t\right) = \mathbb{D} \backslash \mathbb{D}^a\left(t\right)$. É assumido que os dispositivos são voláteis, ou

seja, os dispositivos podem alternar entre os estados *ativo* e *inativo* em qualquer momento e, portanto, um mesmo dispositivo $d \in \mathbb{D}^a(t')$ pode pertencer a $\mathbb{D}^i(t''), t' \neq t''$.

Seja o serviço demandado pelos clientes de um provedor de um sistema OddCI definido por uma sequência de tuplas r_1, r_2, ..., r_n com $r_j =< t_j, q_j, l_j >$, onde t_j é o momento no qual r_j é submetida, q_j é a quantidade desejada de dispositivos simultâneos que devem ser alocados e l_j é a duração do intervalo de tempo no qual os q_j recursos serão necessários ($t_j, q_j, l_j \in \mathbb{N}$). A instância OddCI $\mathbf{I}_j$, $1 \leq j \leq n$, representa o atendimento da requisição r_j pelo sistema.

Seja $L(d,t)$ a função boleana que indica se o dispositivo d está alocado a alguma instância no tempo t, o conjunto $\mathbb{D}^a(t)$ pode ser decomposto em $\mathbb{D}^a(t) = \mathbb{D}^l(t) \cup \mathbb{D}^d(t)$, onde $\mathbb{D}^l(t)$ é o subconjunto dos dispositivos ativos e alocados a instâncias no momento t ($\mathbb{D}^l(t) = \{d \mid d \in \mathbb{D}^a(t) \bigwedge L(d,t) = true\}$) e $\mathbb{D}^d(t)$ é o subconjunto dos dispositivos ativos que estão disponíveis no momento t ($\mathbb{D}^d = \mathbb{D}^a(t) \setminus \mathbb{D}^l(t)$).

Um controlador ao ser designado pelo provedor, através de uma estratégia de escalonamento, para o atendimento de uma demanda r_j, tentará fazer a alocação dos q_j dispositivos solicitados através do envio de mensagens de controle para a rede de *broadcast* que controla. Seja m uma mensagem de controle enviada através do canal unidirecional no momento t, então todos os dispositivos pertencentes a $\mathbb{D}^d(t + T(m))$ receberão e processarão m, onde $T(m)$ é a duração da transmissão da mensagem de controle m. $T(m)$ é uma função da taxa de transmissão e do retardo médio do canal unidirecional e do tamanho da mensagem m.

Seja $\mathbb{D}^r(m) \subseteq \mathbb{D}^d(t + T(m))$ o subconjunto dos dispositivos ativos *disponíveis* em $t + T(m)$ que responderem, através dos seus respectivos canais bidirecionais, à convocação do controlador feita pela mensagem m. O subconjunto $\mathbb{D}^v(m)$ com os primeiros q_j dispositivos de $\mathbb{D}^r(m)$ que atendam a um critério de elegibilidade serão alocados para a instância $\mathbf{I}_j$. Os demais dispositivos, $\mathbb{D}^r(m) \setminus \mathbb{D}^v(m)$, serão descartados.

Para lidar com a volatilidade do sistema, assumimos que o sistema de tarifação adotado pelo provedor pelo uso de seus recursos é baseado na apuração de cada intervalo de tempo com duração σ, chamado *slot* de processamento, durante o qual um dispositivo permanece ativo e alocado a uma instância. Sempre que um dispositivo d é alocado para a instância $\mathbf{I}_j$ em um momento t, o *slot* de processamento $s_{j,d,t}$ é iniciado. O *slot* $s_{j,d,t}$ é dito completado se d permanece alocado para a instância $\mathbf{I}_j$ até o momento $t + \sigma$. Apenas *slots* completados

são tarifados.

Seja $\mathbb{S}_j^i$ o conjunto de todos os *slots* iniciados na instância $\mathbf{I}_j$ e seja $O\left(j,d,t\right)$ uma função boleana que indica se o *slot* $s_{j,d,t}$ foi completado, então o conjunto de *slots* completados na instância $\mathbf{I}_j$ é dado por $\mathbb{S}_j^c = \left\{s_{j,d,t} \mid s_{j,d,t} \in \mathbb{S}_j^i \bigwedge O\left(j,d,t\right) = true\right\}$. Uma instância $\mathbf{I}_j$ é completada quando um mínimo de $\left\lceil \frac{l_j}{\sigma} \right\rceil \times q_j$ *slots* de processamento *completados* é atingido, ou seja, $\left|\mathbb{S}_j^c\right| = \left\lceil \frac{l_j}{\sigma} \right\rceil \times q_j$. Caso $\mathbf{I}_j$ ainda não tenha sido completada quando o *slot* $s_{j,d,t}$ for completado, o dispositivo d será realocado à instância $\mathbf{I}_j$, iniciando o *slot* $s_{j,d,t+\sigma}$. Note que, eventualmente, *slots* adicionais podem ser finalizados após a instância ter sido finalizada.

Seja $I\left(d,t\right)$ a função que indica a qual instância o dispositivo $d \in \mathbb{D}^a\left(t\right)$ está alocado com exclusividade no tempo t:

$$I\left(d,t\right) = \begin{cases} j, \text{ se } \textbf{\textit{d}} \text{ está alocado à instância } \textbf{\textit{I}}_{\textbf{\textit{j}}} \text{ no momento } \textbf{\textit{t}} \\ 0, \text{ se } \textbf{\textit{d}} \text{ não está alocado em nenhuma instância no momento } \textbf{\textit{t}} \end{cases}, \; d \in \mathbb{D}^a\left(t\right),$$

então o conjunto de dispositivos alocados à instância $\mathbf{I}_j$ no momento t, $\mathbb{D}_j^l\left(t\right)$, é dado por $\mathbb{D}_j^l\left(t\right) = \left\{d \mid d \in \mathbb{D}^a\left(t\right) \bigwedge I\left(d,t\right) = j\right\}$.

5.5.2 O Desafio da Alta Volatilidade

Como é asumido que os dispositivos acessíveis pela rede de *broadcast* são voláteis, os dispositivos ativos alocados à instância $\mathbf{I}_j$ podem, eventualmente, se tornar inativos em qualquer momento e tais perdas de dispositivos precisam ser identificadas e repostas.

A reposição de dispositivos para a instância $\mathbf{I}_j$ no momento t através do envio de uma mensagem de controle m levará o tempo $T\left(m\right)$ para atingir os dispositivos ativos disponíveis no momento $t + T\left(m\right)$, $\mathbb{D}^d\left(t + T\left(m\right)\right)$. Neste sentido, a estratégia de provisionamento adotada pelo controlador precisa considerar a reposição tanto dos dispositivos já perdidos por $\mathbf{I}_j$ no momento t, quanto dos que poderão ser perdidos adicionalmente até o momento $t + T\left(m\right)$.

Além disso, a quantidade de dispositivos que responderem à mensagem de controle m, $|\mathbb{D}^r\left(m\right)|$, deve ser o mais próximo possível da quantidade de dispositivos que serão alocados a $\mathbf{I}_j$ em decorrência do envio de m, $|\mathbb{D}^v\left(m\right)|$. Para tal, o cálculo de $P\left(m\right)$, que representa a probabilidade de cada dispositivo em $\mathbb{D}^d\left(t + T\left(m\right)\right)$, responder ou não à mensagem m enviada no momento t, deve levar em consideração a quantidade de dis-

positivos que se necessita e a quantidade total de dispositivos que estarão disponíveis: $P(m) = |\mathbb{D}^v(m)|/|\mathbb{D}^d(t+T(m))|$. Neste sentido, como o estado dos dispositivos da rede de *broadcast* pode mudar constantemente, é necessário dispor de algum mecanismo para fazer, em t, uma estimativa do número de dispositivos disponíveis em um momento futuro, $t+T(m)$.

Por outro lado, para minimizar a perda de dispositivos em $\mathbf{I}_j$, o controlador precisa adotar algum critério de elegibilidade para indicar, dentre os dispositivos existentes em $\mathbb{D}^d(t+T(m))$ que irão responder a m, aqueles dispositivos que possuam uma expectativa de maior permanência no estado ativo.

Do ponto de vista do cliente, a existência da volatilidade do sistema implica na necessidade de adequar o tamanho máximo das tarefas da sua aplicação como um divisor do tamanho do *slot* de processamento adotado pelo provedor, ou seja, deve ser possível a conclusão total ou parcial (via *checkpoints*) de uma ou mais tarefas durante a duração de um *slot* de processamento.

5.5.3 Descrição dos Experimentos

Para analisar como a volatilidade e a contenção de recursos presentes na rede de *broadcast* podem afetar a disponibilidade coletiva necessária, foram considerados dois cenários de uso:

- *Atendendo a Aplicações Sensíveis ao Tempo*: No primeiro cenário, chamado *Vazão Mínima*, o controlador tenta garantir que a duração esperada para a instância $\mathbf{I}_j$ seja observada, ou seja, que os $\left\lceil \frac{l_j}{\sigma} \right\rceil \times q_j$ *slots* solicitados sejam completados no tempo l_j. Uma das formas de conseguir isso é fazer com que o número de *slots* completados na instância $\mathbf{I}_j$ permaneça em um valor médio que seja maior ou igual a q_j durante todo o ciclo de vida de $\mathbf{I}_j$. Para lidar com a eventual perda de dispositivos e mesmo assim garantir uma vazão mínima q_j, o controlador deve aplicar um determinado nível de redundância sobre o tamanho mínimo desejado para a instância. Para isso, são enviadas, proativamente, mensagens de controle para regenerar o tamanho da instância para um valor alvo $q_j + X$, onde X é a quantidade adicional necessária para compensar as eventuais perdas de dispositivos que ocorrerão até o envio do próximo comando de regeneração. Baseado na última consolidação de *heartbeat messages*, o controlador

calcula X, o momento t para envio de cada mensagem de controle m para a instância $\mathbf{I}_j$ e também $|\mathbb{D}^d\left(t+T\left(m\right)\right)|$ em função da taxa histórica de perda de dispositivos observada na rede de *broadcast* em um dado período de referência, cujo momento inicial padrão é o momento de submissão da demanda r_j, ou seja, t_j. O valor $P\left(m\right)$ é definido pelo controlador para cada mensagem de controle m considerando q_j, X, $|\mathbb{D}_j^l(t)|$ e $|\mathbb{D}^d\left(t+T\left(m\right)\right)|$ da seguinte forma: $P\left(m\right)=((q_j+X)-|\mathbb{D}_j^l(t)|)/|\mathbb{D}^d\left(t+T\left(m\right)\right)|$. Neste cenário, é aceitável que o tamanho solicitado para a instância (q_j) seja excedido para compensar regimes de maior volatilidade.

- *Lidando com Capacidade Limitada no Backend*: No segundo cenário, chamado *Paralelismo Máximo*, o controlador tenta cumprir, tanto quanto possível, o limite do tamanho q_j solicitado para a instância sem excedê-lo. Assim, o número de dispositivos alocados para a instância $\mathbf{I}_j$, tende a permanecer em uma quantidade sempre igual ou menor do que q_j durante todo o seu ciclo de vida para respeitar a condição de que o *Backend* do cliente só consegue tratar, no máximo, q_j dispositivos simultaneamente. Sempre que a perda de dispositivos causada pela volatilidade da rede de *broadcast* atingir um determinado limite Y, ou seja, $\left|\mathbb{D}_j^l\left(t\right)\right| \leq q_j - Y$, serão enviadas, reativamente, mensagens de controle para regenerar o tamanho da instância para o valor alvo q_j. O valor adequado de Y, que representa o tempo de reação para regeneração da instância, e é definido pelo controlador a partir do tempo $T\left(m\right)$ necessário para transmissão da mensagem de controle m, bem como em função da taxa histórica de perda de dispositivos observada na rede de *broadcast*. O valor $P\left(m\right)$ é definido pelo controlador para cada mensagem de controle m considerando q_j, Y, $|\mathbb{D}_j^l(t)|$ e $|\mathbb{D}^d\left(t+T\left(m\right)\right)|$ da seguinte maneira: $P\left(m\right)=max(q_j-|\mathbb{D}_j^l(t)|,Y)/|\mathbb{D}^d\left(t+T\left(m\right)\right)|$. Neste cenário, é aceitável que a duração solicitada (l_j) não seja cumprida em regimes de maior volatilidade.

Implementação do Modelo de Simulação

O simulador usado nos experimentos, chamado OddCISim foi baseado no ambiente OMNeT++ [Varga e Hornig 2008], uma biblioteca e *framework* de simulação modular e baseado em componentes, que pode ser estendido usando a linguagem C++ para a lógica dos com-

ponentes, enquanto que a linguagem *NEtwork Description* (NED) é usada para descrição da topologia da rede, portas de comunicação, canais, conexões, dentre outros parâmetros. Para essa avaliação, algumas extensões nos componentes originais foram realizadas. Em particular, foram acrescentados os aspectos de transmissão em *broadcast* e o comportamento dos componentes da arquitetura, de acordo com o modelo de operação descrito na Seção 5.2.1 e o modelo de simulação e cenários de uso descritos nas Seções 5.5.1 e 5.5.3, respectivamente[10].

Parte da configuração do simulador foi baseada em outra etapa da pesquisa na qual foram obtidas medições de campo em um *testbed* real: um protótipo de sistema OddCI para redes de TV Digital [Costa et al. 2012c], cujos resultados, descritos no Capítulo 6, permitiram confirmar o comportamento linear na transmissão de mensagens de controle por radiodifusão, a adequação dos recursos de comunicação direta dos receptores para troca de tarefas/resultados e o potencial de processamento de receptores de baixo custo (*low-end*).

O comportamento estocástico do sistema OddCI simulado foi modelado usando algumas variáveis independentes (aleatórias). A população de dispositivos computacionais (ou nós) potencialmente acessíveis através da rede de *broadcast*, representada pelo conjunto $\mathbb{D}$, é determinada, *a priori*, como um parâmetro de simulação. Entretanto, a quantidade de nós ativos (i.e, que podem ser efetivamente atingidos por uma mensagem de controle) no início da simulação é modelada como uma variável aleatória com distribuição uniforme: $|\mathbb{D}^a(0)| = U(\mu, |\mathbb{D}|)$, onde μ é o número mínimo de dispositivos acessíveis através da rede de *broadcast*. Uma vez que o número inicial de dispositivos ativos $|\mathbb{D}^a(0)|$ é determinado no início da simulação, os dispositivos ativos iniciais são selecionados entre a população de dispositivos, $\mathbb{D}$, com igual probabilidade. Sempre que um nó individual é selecionado para ser ativado, ele permanece ativo por um tempo de sessão τ_{ON} e então é desativado por um período de espera (*standby*) τ_{OFF}. Dessa forma, os dispositivos ativos em um determinado momento na rede de *broadcast* configuram um processo estocástico que depende das seguintes variáveis: tamanho da população $|\mathbb{D}|$, o número inicial de dispositivos ativos, $|\mathbb{D}^a(0)|$, o tempo em sessão, τ_{ON}, e o tempo em *standby*, τ_{OFF}. Foi assumido um mesmo *ranking* de disponibilidade para os dispositivos em $\mathbb{D}$.

A volatilidade ($\mathcal{V}$) inserida no sistema simulado foi normalizada, através das probabi-

[10]O modelo completo do simulador usado neste trabalho pode ser encontrado no sítio http://www.lsd.ufcg.edu.br/∼rostand/JiTDC_OddCISim.zip.

lidades utilizadas em τ_{ON} e τ_{OFF} (que foram modeladas como variáveis aleatórias com distribuição Bernoulli), de forma a obter uma variação percentual controlada da quantidade de dispositivos que alternam entre o estado ativo e inativo na rede de *broadcast* dentro de cada período de tempo de tamanho σ, o intervalo de referência considerado, mas mantendo o total de ativos em qualquer tempo próximo da disponibilidade inicial configurada. Em resumo, o parâmetro $\mathcal{V}$ regula o percentual de dispositivos ativos ganhos e perdidos em um dado intervalo de tempo de tamanho σ, o mesmo adotado como duração de um *slot* de processamento. É possível que esta associação da volatilidade à duração do *slot* de processamento possa tornar os resultados obtidos na configuração estudada potencialmente aplicáveis em outros cenários de tarifação e granularidade de tarefas.

Para analisar o comportamento do sistema sob alta volatilidade em regimes de contenção de recursos, a carga de trabalho utilizada teve como objetivo estressar dois gargalos potenciais: a disponibilidade de dispositivos para atendimento da demanda e a concorrência pelo uso do canal de transmissão em *broadcast*. Para tal, foi fixado um pico de demanda ($\mathcal{P}$), representando o máximo da soma de dispositivos alocados para instâncias em um dado momento de um período de observação. A partir de $\mathcal{P}$, as cargas de trabalho de cada experimento foram construídas de forma relativa usando dois parâmetros do simulador: quantidade de instâncias simultâneas ($\mathcal{S}$) e a duração das instâncias em *slots* ($\mathcal{D}$). Assim, o *workload* de cada experimento é baseado na sua configuração e formado por $\mathcal{S}$ instâncias simultâneas iguais, todas iniciando no mesmo momento ($t_j = 0$), solicitando a mesma quantidade de dispositivos ($q_j = \frac{\mathcal{P}}{\mathcal{S}}$) pelo mesmo intervalo de tempo ($l_j = \mathcal{D} \times \sigma$). O tamanho de $\mathbb{D}$ é regulado pela aplicação de um *fator de contenção*, ζ, sobre $\mathcal{P}$: $|\mathbb{D}| = \zeta \times \mathcal{P}$.

Parâmetros do Sistema

Para atribuição dos parâmetros do sistema foram usadas duas estratégias: projeto de experimento (DoE) e varredura de parâmetros. Inicialmente, os parâmetros foram tratados em cada cenário considerado através de um DoE do tipo 2^k fatorial [Jain 1991].

Os fatores considerados no DoE foram: *Volatilidade* ($\mathcal{V}$), *Tamanho da População* ($|\mathbb{D}|$), *Tamanho da Imagem* ($\mathcal{T}$), *Instâncias Simultâneas* ($\mathcal{S}$) e *Duração da Instância* ($\mathcal{D}$).

Para o tamanho da imagem da aplicação, o qual está associado ao tempo de uso do canal de transmissão em *broadcast* para envio de cada mensagem de controle, foram considera-

Tabela 5.4: DoE 2^k: Fatores, níveis e efeitos para o cenário *Vazão Mínima*

Fator	**Baixo**	**Alto**	**Efeito Estimado**	**Soma dos Quadrados**	**Contribuição**
A: Volatilidade ($\mathcal{V}$)	5%	75%	$-0,33$	$0,89$	$28,41\%$
B: População ($\|\mathbb{D}\|$)	$(1+\mathcal{V})\times\mathcal{P}$	$10\times\mathcal{P}$	$0,27$	$0,57$	$18,24\%$
C: Tamanho da Imagem ($\mathcal{T}$)	$512Kb$	$5Mb$	$-0,17$	$0,22$	$7,10\%$
D: Instâncias Simultâneas ($\mathcal{S}$)	10	100	$-0,17$	$0,24$	$7,64\%$
E: Duração da Instância ($\mathcal{D}$)	$10\ horas$	$100\ horas$	$-0,02$	$0,01$	$0,09\%$

dos dois valores diferentes: *pequeno* (representativo do tamanho de módulos clientes de aplicações como o SETI@home [Anderson et al. 2002] e *grande* (representando "*workers*" de implementações padrão de *desktop grids* como o OurGrid [Cirne et al. 2006]). As imagens do tipo *pequeno* têm 512 Kbytes de tamanho, enquanto que as imagens do tipo *grande* possuem tamanho de 5 Mbytes. Os níveis atribuídos para os demais fatores em cada DoE estão apresentados nas Tabelas 5.4 e 5.5.

A variável de resposta considerada para o cenário do *Vazão Mínima* foi o *coeficiente médio de vazão* ($\overline{\Phi}$) das instâncias, o qual representa a relação entre a quantidade média de *slots* completados por ciclo e a quantidade necessária para que a duração esperada para a instância seja cumprida. Essa métrica é dada por $\overline{\Phi} = (\sum_{j=1}^{\mathcal{S}}(|\mathbb{S}_j^c|/\mathcal{D}/q_j))/\mathcal{S}$ e seu valor de referência é 1.

Para o cenário do *Paralelismo Máximo* foi escolhida a variável de resposta *coeficiente médio de paralelismo* ($\overline{\Pi}$) das instâncias, o qual representa a relação entre a quantidade efetiva de dispositivos fornecida e a quantidade de dispositivos solicitada. Esta métrica é dada por $\overline{\Pi} = (\sum_{j=1}^{\mathcal{S}}(\overline{|\mathbb{D}_j^l|}/q_j))/\mathcal{S}$ e seu valor de referência também é 1.

Foram conduzidas várias repetições dos 32 experimentos previstos no DoE realizado para cada um dos cenários considerados para obter médias com intervalos de confiança de 95%. A contribuição de cada fator em cada cenário é mostrada nas Tabelas 5.4 (*Vazão Mínima*) e 5.5 (*Paralelismo Máximo*).

No cenário de *Vazão Mínima*, os fatores da *Volatilidade* e do *Tamanho da População* foram preponderantes com participação de $28,41\%$ e $18,24\%$, respectivamente (Tabela 5.4). Enquanto que no cenário de *Paralelismo Máximo*, além da *Volatilidade*, que responde por

Tabela 5.5: DoE 2^k: Fatores, níveis e efeitos para o cenário *Paralelismo Máximo*

Fator	**Baixo**	**Alto**	**Efeito Estimado**	**Soma dos Quadrados**	**Contribuição**
A: Volatilidade ($\mathcal{V}$)	5%	75%	$-0,22$	$0,39$	$16,17\%$
B: População ($\|\mathbb{D}\|$)	$(1+\mathcal{V}) \times \mathcal{P}$	$10 \times \mathcal{P}$	$0,04$	$0,02$	$0,66\%$
C: Tamanho da Imagem ($\mathcal{T}$)	$512Kb$	$5Mb$	$-0,23$	$0,43$	$17,83\%$
D: Instâncias Simultâneas ($\mathcal{S}$)	10	100	$-0,24$	$0,46$	$19,16\%$
E: Duração da Instância ($\mathcal{D}$)	$10\ horas$	$100\ horas$	$0,01$	$0,00$	$0,02\%$

$16,17\%$, os fatores *Tamanho da Imagem* com $17,83\%$ e *Instâncias Simultâneas* com $19,16\%$ foram determinantes na variação da métrica observada (Tabela 5.5).

Como resultado da análise dos efeitos através de ANOVA [Jain 1991], o F-Value de $164,4793$ (*Vazão Mínima*) e $252,9781$ (*Paralelismo Máximo*) implicam que os modelos são significativos. O R^2 ajustado indica que os modelos explicam $98,75\%$ e $98,27\%$ da variação observada e o R^2 de predição está dentro de $0,20$ do R^2 ajustado, representando uma boa capacidade de predição dos modelos [11].

Para a realização das simulações, os valores dos parâmetros que não afetaram o comportamento da variável de resposta foram ajustados para os valores médios entre os níveis "Alto" e "Baixo" usados em cada DoE[12]. Para os fatores mais relevantes: *Volatilidade* e *Tamanho da População* (*Vazão Máxima*) e *Volatilidade*, *Tamanho da Imagem* e *Instâncias Simultâneas* (*Paralelismo Máximo*), foi aplicada uma varredura de parâmetros. Para a varredura não foi necessário ampliar os níveis usados no DoE, posto que já ocorreram restrições relevantes nos respectivos intervalos.

A Tabela 5.6 mostra como o sistema foi configurado para os experimentos dos dois cenários, usando o resultado do DoE, os valores obtidos no *testbed* real e alguns padrões de mercado, como no caso da duração do *slot* de processamento baseada na mesma forma de tarifação usada nas *spot instances* da AWS.

[11] Maiores detalhes sobre este estudo, incluindo os gráficos de diagnóstico, cubo e interação, podem ser encontrados no projeto Möbius [Deavours et al. 2002] que está disponível *online* em http://www.lsd.ufcg.edu.br/~rostand/JiTDC_OddCISimDoE.zip.

[12] Exceto no caso da *Duração da Instância*, com contribuição irrelevante, onde foi usado o nível "Baixo" com o objetivo de diminuir o tempo de execução de cada experimento.

Tabela 5.6: Parâmetros Usados nas Simulações

Parâmetro	**Cenário *Vazão Mínima***	**Cenário *Paralelismo Máximo***
Pico de Demanda ($\mathcal{P}$)	10.000 dispositivos	10.000 dispositivos
Taxa Canal Direto	1 Mbps	1 Mbps
Taxa Canal de *Broadcast*	1 Mbps	1 Mbps
Duração *slot* de processamento (σ)	1 hora	1 hora
Retardo Máximo	5 segundos	5 segundos
Disponibilidade Inicial ($\|\mathbb{D}^a(0)\|$)	100% da população	100% da população
Duração da Instância ($\mathcal{D}$)	10 *slots*	10 *slots*
Instâncias Simultâneas ($\mathcal{S}$)	50 instâncias	{20,40,60,80} instâncias
Tamanho da Imagem ($\mathcal{T}$)	$\{2, 5\}$ MB	{1MB,2MB,3MB,4MB}
População ($\|\mathbb{D}\|$)	$\{2.\mathcal{P}$,$3.\mathcal{P}$,$4.\mathcal{P}$,$5.\mathcal{P}$, $6.\mathcal{P}$,$7.\mathcal{P}$, $8.\mathcal{P}$,$9.\mathcal{P}\}$	$10.\mathcal{P}$
Volatilidade ($\mathcal{V}$)	{20%,30%,40%,50%, 60%,70%,80%,90%}	{20%,30%,40%,50%, 60%,70%,80%,90%}

Validação e Verificação

Pelo fato do modelo conceitual de um sistema OddCI representar uma arquiteura nova, sem correspondência no mundo real, uma validação do mesmo não se aplica. Mas nós fizemos uma série de atividades de verificação no sentido de assegurar que a implementação do modelo conceitual foi feita de forma correta.

A primeira técnica utilizada foi a animação. Usando os recursos de animação do ambiente OMNeT++ foi possível acompanhar visualmente o comportamento operacional das entidades do modelo ao longo do tempo, permitindo verificar se as interações entre os diversos componentes da arquitetura ocorria de forma tempestiva e ordenada.

A segunda atividade de verificação baseou-se na construção de gráficos operacionais com as saídas do modelo para observar se as métricas obtidas, com seus respectivos indicadores de desempenho, estavam em sintonia com a lógica do modelo e apresentavam a acurácia desejada.

Em seguida, com a escolha apropriada dos parâmetros de configuração, foram realizados *testes degenerados* e *testes de condição extrema* para verificação do comportamento do modelo de simulação em cenários especiais. O objetivo aqui foi observar se a estrutura e

as saídas do modelo se apresentavam de forma plausível mesmo quando expostas a uma combinação extrema de valores de parâmetros. A Tabela 5.7 traz um resumo dos testes realizados, os quais foram aplicados para os dois cenários de uso considerados com resultados similares e dentro do comportamento esperado. Os testes foram repetidos para a produção de instâncias com um total de 1.000 e $1.000.000$ de *slots*.

Também foi feita uma verificação das adaptações introduzidas no OMNeT++ e a consistência das saídas do simulador foi exaustivamente verificada, tanto com relação à adequação das respostas para as combinações de parâmetros de configuração, quanto com relação ao estado interno das variáveis do simulador em cada momento do período de observação. Uma trilha de auditoria (*traços*) com registros exclusivos foi criada apenas para subsidiar esta fase de verificação. Além de testes de aceitação, a análise dos traços permitiu verificar a *validade aparente* do modelo, ou seja, se o mesmo representa de forma adequada a arquitetura proposta, e também a sua *validade de eventos*, aferida através de rastreamento dos eventos associados com os componentes principais que ocorreram nas simulações para verificar a sua compatibilidade com os eventos esperados no modelo. Em especial, foi cuidadosamente observado se as ações dos mecanismos compensatórios do *Controller* eram disparadas corretamente, em termos de tempestividade e de precisão, em resposta às variações de tamanho das instâncias causadas por mudanças no estado da rede de *broadcast* nos diversos cenários de volatilidade simulados.

5.5.4 Resultados e Análise

No primeiro experimento, realizado para o cenário de *Paralelismo Máximo*, o objetivo foi observar como a variação da volatilidade ($\mathcal{V}$), da quantidade de instâncias simultâneas ($\mathcal{S}$) e do tamanho da imagem da aplicação ($\mathcal{T}$) impacta na manutenção da quantidade desejada de dispositivos ativos para cada instância. Para eliminar a variável de contenção de dispositivos, a população foi configurada para 10 vezes o total da demanda concomitante esperada ($|\mathbb{D}| = 10 \times \mathcal{P}$). Para cobrir todas as combinações dos parâmetros de entrada foram realizados 128 experimentos - repetidos até que as médias obtidas tivessem intervalo de confiança de 95%. A métrica de interesse observada foi o coeficiente médio de paralelismo das instâncias, $\overline{\Pi}$. Os resultados obtidos estão exibidos graficamente nas figuras 5.5 e 5.6.

Como pode ser observado na Fig. 5.5(a), quando lida com imagens de aplicação peque-

Tabela 5.7: Testes degenerados e de condição extrema do simulador OddCISim

Teste	**Tamanho da População**	**Volatilidade Inserida**	**Disponibilidade Inicial**	**Resultado Observado**
1	0	0%	0%	Foram enviadas diversas WMs mas não houve retorno para alocação por parte de dispositivos ativos. Por não haver dispositivos ativos nenhuma instância foi instanciada.
2	$\mathcal{P}$	0%	0%	O resultado obtido foi idêntico ao do teste #1.
3	$10.\mathcal{P}$	0%	0%	O resultado obtido foi idêntico ao do teste #1.
4	0	0%	100%	Por não haver nenhum dispositivo na rede de *broadcast*, o resultado obtido também foi idêntico ao do teste #1.
5	$\mathcal{P}$	0%	100%	Sem volatilidade e com a quantidade de recursos exata equivalente ao pico de demanda da carga de trabalho utilizada, as instâncias foram completadas com resultado ótimo: instanciadas com apenas uma WM e completadas no tempo mínimo.
6	$10.\mathcal{P}$	0%	100%	O resultado obtido foi idêntico ao do teste #5. A maior quantidade de recursos disponíveis na rede de *broadcast* não fez diferença nessa configuração.
7	0	100%	0%	A inserção de volatilidade se comportou exatamente como modelado, mantendo uma relação constante entre a quantidade de dispositivos que alternam entre o estado ativo e inativo. Como a disponibilidade inicial era de nenhum dispositivo ativo, este quadro se manteve durante o período de observação levando à um resultado similar ao do teste #1.
8	$\mathcal{P}$	100%	0%	O resultado obtido foi idêntico ao do teste #7.
9	$10.\mathcal{P}$	100%	0%	O resultado obtido foi idêntico ao do teste #7.
10	0	100%	100%	Oresultado obtido foi idêntico ao do teste #1.
11	$\mathcal{P}$	100%	100%	Neste teste, as instâncias foram criadas mas apresentaram uma vazão muito baixa e demandaram mais de 30 vezes o tempo mínimo para serem finalizadas. A baixa disponibilidade de recursos impediu a aplicação dos níveis de redundância necessários, apesar da volatilidade do sistema ter sido bem estimada pelo *Controller*.
12	$10.\mathcal{P}$	100%	100%	Com mais recursos disponíveis, a vazão foi melhorada pela aplicação de maior redundância e as instâncias foram finalizadas em um terço do tempo obtido no teste #11.

nas, o controlador consegue compensar a perda de dispositivos em praticamente todos os regimes de volatilidade simulados, mesmo quando coordenando muitas instâncias simultâneas. Entretanto, à medida que o tamanho da imagem aumenta, aumenta o tamanho da mensagem de controle correspondente e diminui a capacidade do controlador de restabelecer o nível de paralelismo máximo das instâncias devido ao aumento proporcional do tempo de transmissão de cada mensagem de controle (Fig. 5.5(b)). Isso fica ainda mais evidenciado com o incremento no número de instâncias simultâneas, o que implica, na prática, no enfileiramento de mensagens de controle para serem enviadas pelo transmissor de *broadcast*. Esse efeito, que pode ser visualizado também nas figuras 5.6(a) e Fig. 5.6(b), é ampliado pelas restrições ao paralelismo máximo impostas neste cenário de uso, que ao limitar o tamanho que pode ser praticado para cada instância, não permite uma compensação antecipada das perdas através de redundância, o que diminuiria a quantidade de mensagens de controle reparatórias a serem enviadas e, consequentemente, a concorrência das instâncias pelo canal de *broadcast*. Associadamente, a inclusão de mecanismos adequados no controle de admissão pode otimizar o uso dos recursos do sistema através de um melhor escalonamento das instâncias ao longo do tempo.

No segundo experimento, realizado para o cenário de *Vazão Mínima*, o objetivo foi observar como a variação da volatilidade ($\mathcal{V}$) e do tamanho da população de dispositivos ($|\mathbb{D}|$) impactam na manutenção da quantidade desejada de *slots* de processamento completados, ou vazão, obtida em cada instância. Para controlar o nível de contenção de recursos, o tamanho da população foi iniciada em um patamar operacional mínimo, correspondente ao pico da demanda esperada acrescido da volatilidade inserida ($|\mathbb{D}| = \mathcal{P} \times (1 + V)$), e foi sendo aumentada pela aplicação de um fator de contenção (um fator 2 equivale a uma população com o dobro da quantidade operacional mínima, um fator 3, ao triplo, e assim por diante). Para cobrir todas as combinações dos parâmetros de entrada foram realizados 64 experimentos - repetidos até que as médias obtidas tivessem intervalo de confiança de 95%. A métrica de interesse principal foi a mesma usada no DoE, o coeficiente médio de vazão das instâncias, $\overline{\Phi}$. Os resultados obtidos estão exibidos na figuras 5.7 e 5.8.

Como ilustrado na Fig. 5.7(a), a quantidade média de *slots* de processamento completados por ciclo é fortemente afetada à medida que é inserida mais volatilidade no sistema. Nas configurações com até 40% de volatilidade, ou seja, onde até 40% dos dispositivos alocados

às instâncias falham em cada ciclo, foi possível manter níveis de vazão apenas 10% abaixo do solicitado, dependendo do fator de contenção do tamanho da população aplicado. Em tais níveis de volatilidade, o esforço de coordenação do provedor também é mantido controlado, como pode ser visto na Fig. 5.7(b), a qual traz o percentual de *slots* iniciados que não foram completados. Entretanto, à medida que a volatilidade é incrementada, a vazão entregue diminuiu consideravelmente apesar do aumento do custo operacional do provedor, com perdas de até 90% para a obtenção de vazão de apenas 30%. Cada *slot* não finalizado implica em custos operacionais, diretos e indiretos, para o provedor, principalmente no consumo de recursos de comunicação via canal de *broadcast* e canal direto dos dispositivos.

A métrica coeficiente médio de paralelismo das instâncias, $\overline{\Pi}$, também foi apurada para esse experimento. Pode ser visualizado na Fig. 5.8(a) que, por não haver restrição de tamanho para as instâncias, a quantidade de dispositivos ativos nas instâncias foi sendo aumentada à medida que a volatilidade percebida no sistema aumentava e ainda havia disponibilidade de recursos. O resultado do aumento do paralelismo repercute em uma atenuação dos efeitos da volatilidade sobre a vazão, como pode ser visualizado na Fig. 5.8(b), na qual a duração das instâncias torna a diminuir nos cenários com menor contenção de recursos mesmo em regimes de maior volatilidade. Obviamente, em contextos cuja disponibilidade de recursos não apresentem restrições ao nível de redundância praticados, como é o caso de redes de TV Digital com milhões de dispositivos, é possível aplicar níveis de paralelismo ainda maiores nas instâncias e ampliar a faixa de volatilidade onde alta vazão pode ser praticada. Entretanto, é necessário concilar o nível de paralelismo com a capacidade do *Backend* e com o custo operacional do provedor.

5.6 Considerações Finais

Com o objetivo de viabilizar o uso de recursos terceirizados de alta granularidade, alta volatilidade e alta dispersão para a construção de *JiT DCs* de alta vazão, nós apresentamos uma arquitetura nova, chamada de *On-demand Distributed Computing Infrastructure* (OddCI). Baseados na operação de infraestruturas computacionais distribuídas construídas sob demanda sobre dispositivos computacionais terceirizados organizados como redes de *broadcast*, nós procuramos demonstrar que os sistemas OddCI são tecnicamente viáveis e apresentam um

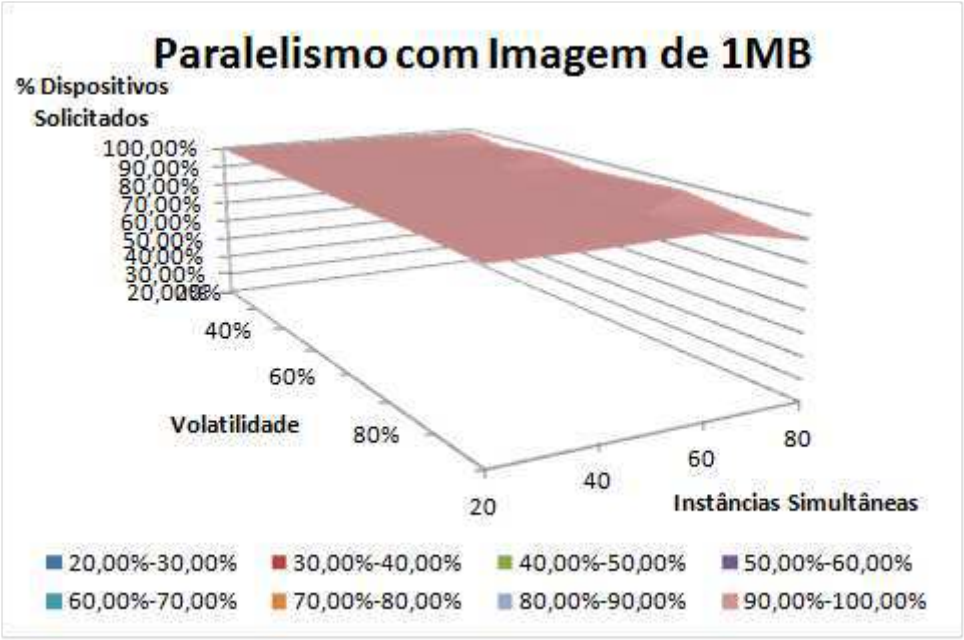
bom potencial para uso em HTC.

Discutimos as questões principais que precisam ser enfrentadas na implementação da arquitetura OddCI proposta, incluindo o esforço de coordenação das instâncias e os aspectos de disponibilidade dos recursos. O comportamento do sistema e o impacto que os seus parâmetros têm sobre a sua eficiência foram cuidadosamente estudados através de experimentos de simulação.

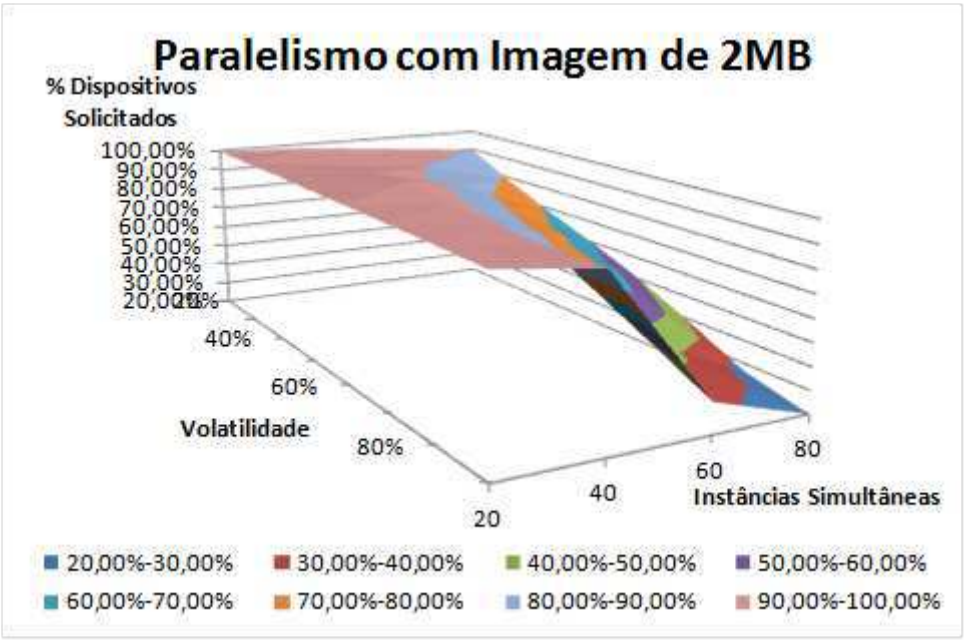
Nossos resultados mostram que, mesmo em cenários de altíssima volatilidade de nós autônomos e distribuídos geograficamente, é possível construir *JiT Clouds* com a disponibilidade coletiva adequada para atingir níveis controlados de vazão computacional usando os mecanismos de coordenação adequados. Entretanto, a viabilidade operacional fica mais evidente nas zonas de volatilidade situadas abaixo dos 40% em ambos os cenários de uso. Acima deste patamar de volatilidade, o nível de redundância necessário para compensar a perda de dispositivos aumenta significativamente o consumo de recursos do sistema. Além disso, a eficiência do sistema também fica mais suscetível à influência de outros fatores como a quantidade de instâncias simultâneas e o nível de contenção da rede de *broadcast* [Costa et al. 2013].

Nós também apresentamos um modelo de segurança para sistemas OddCI em geral que pode ser aplicado na construção de *JiT DCs* de alta vazão voltados para aplicações "best-effort" em geral. Os muitos desafios envolvidos na operação de tais sistemas com base em recursos terceirizados e não dedicados foram levantados e discutidos. Um modelo de segurança baseado em contramedidas adotadas em outros contextos foi proposto para viabilizar a operação adequada de infraestruturas distribuídas e voláteis.

No próximo capítulo, nós iremos investigar o potencial de uso de recursos computacionais terceirizados não convencionais em *JiT DCs* dinâmicos através da abordagem OddCI. Em particular, nós discutiremos como construir um sistema OddCI sobre os recursos de uma rede de TV Digital.

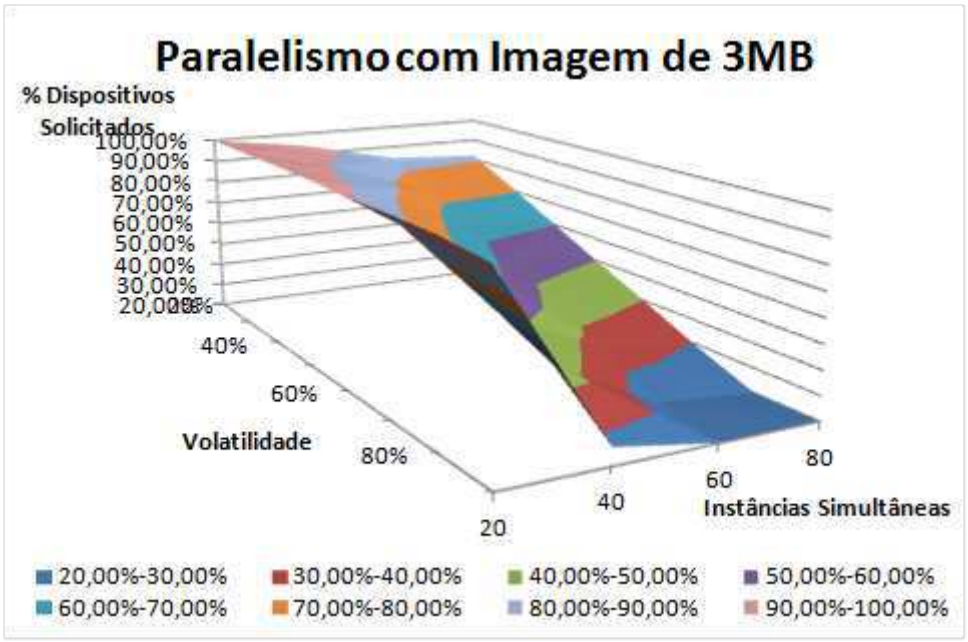


(a)

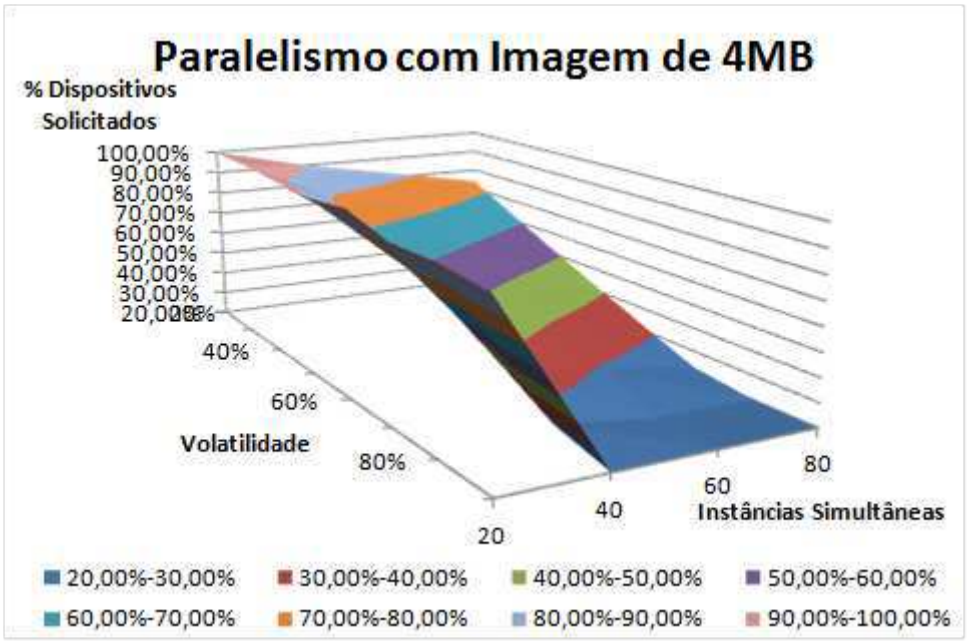


(b)

Figura 5.5: *Paralelismo Máximo*: Métrica $\overline{\Pi}$ para tamanhos de imagens ($\mathcal{T}$) de 1MB e 2Mb

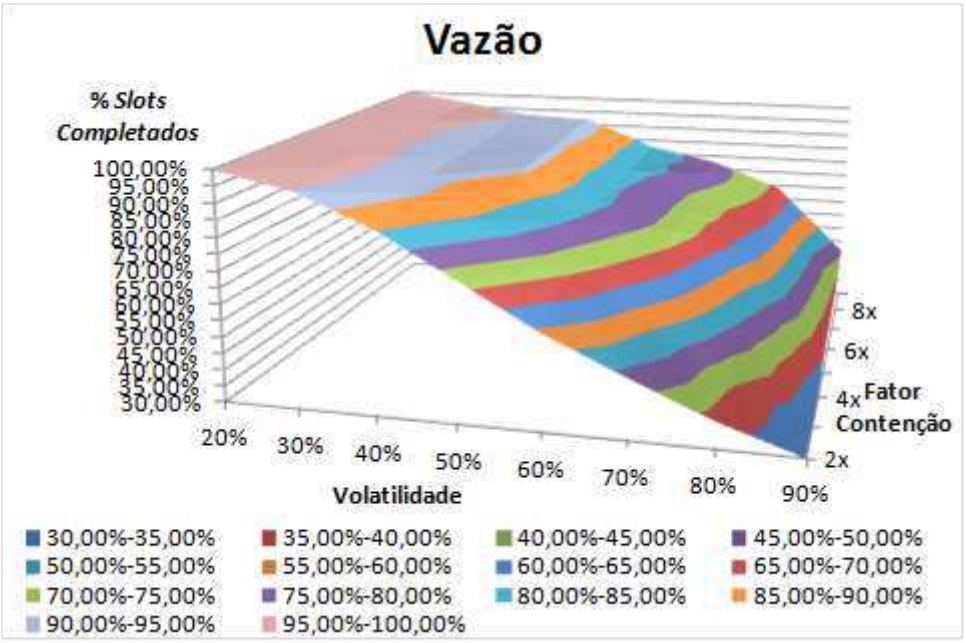


(a)

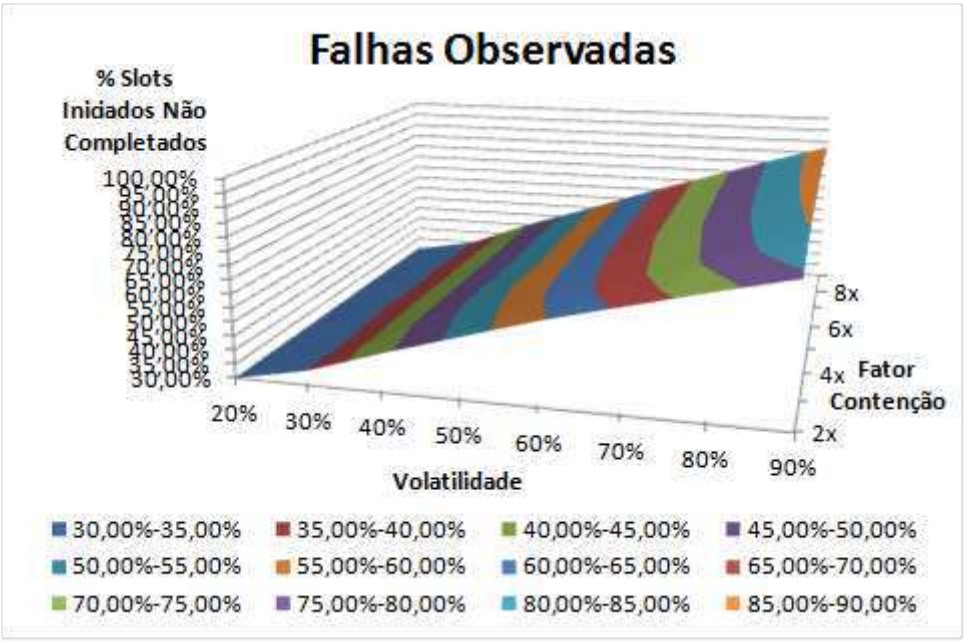


(b)

Figura 5.6: *Paralelismo Máximo*: Métrica $\overline{\Pi}$ para tamanhos de imagens ($\mathcal{T}$) de 3MB e 4Mb

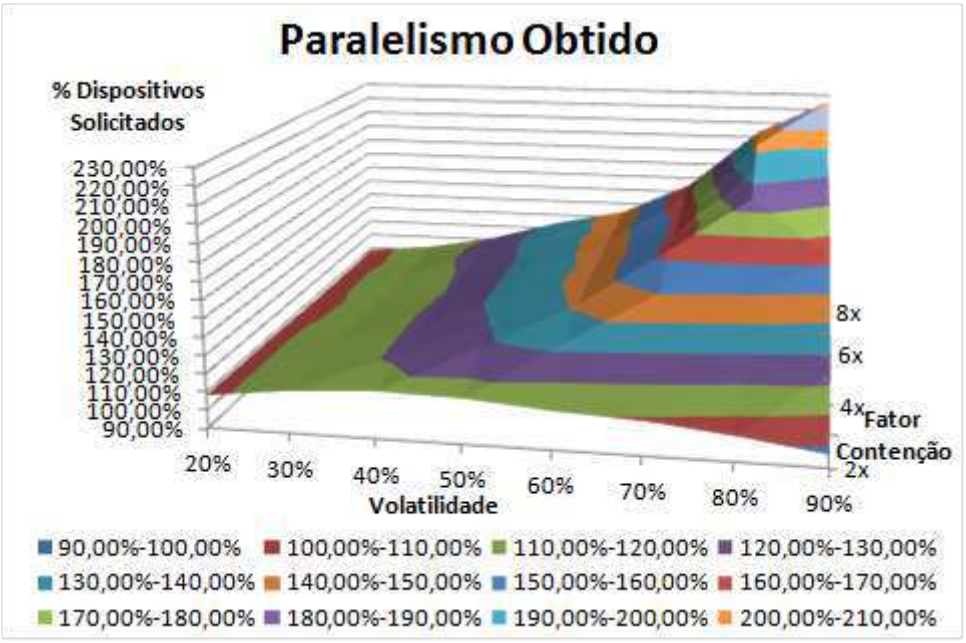


(a)

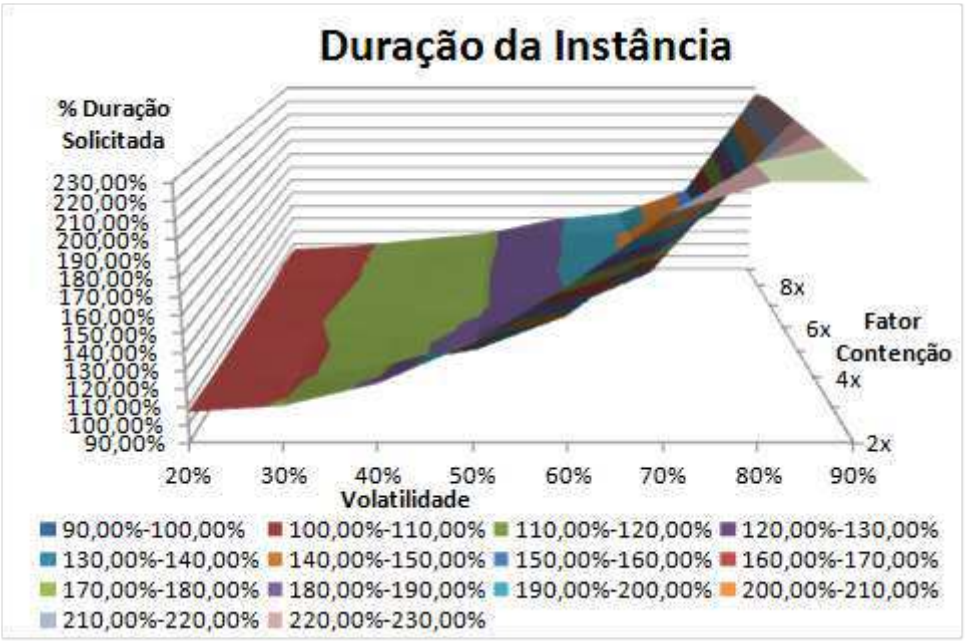


(b)

Figura 5.7: *Vazão Mínima*: Vazão e Falhas Observadas



(a)



(b)

Figura 5.8: *Vazão Mínima*: Paralelismo e Duração da Instância

Capítulo 6

Uso de Recursos Terceirizados Não Convencionais em *JiT DCs* Dinâmicos

A crescente popularidade da Internet a fez extrapolar ambientes acadêmicos, científicos e empresariais e ocupar as residências e o cotidiano das pessoas de uma forma quase que onipresente. Este fenômeno tem trazido a reboque uma série de avanços que estão mudando a forma como computadores são usados hoje em dia. A disponibilidade de acesso a redes de alta velocidade combinada com a crescente oferta de computadores com alta capacidade de processamento, agora cada vez mais acessíveis às camadas da população de mais baixa renda, é um fenômeno em escala mundial.

O cenário tecnológico atual é fortemente orientado para a convergência e marcado pelo surgimento de serviços e dispositivos que combinam tecnologias que surgiram inicialmente em contextos distintos. Desde celulares com capacidade de captura de imagens e vídeo ao provimento de serviços agregados de telefonia, internet e televisão, dos modems móveis para acesso à Internet aos celulares de terceira geração com grande memória e processadores poderosos, praticamente tudo que é digital é potencialmente convergente. Em tal contexto, é possível ampliar as alternativas para além das fronteiras de centros de dados corporativos, passando a considerar também um vasto contingente distribuído de recursos computacionais terceirizados individuais, tanto de natureza convencional, como computadores pessoais, quanto dispositivos computacionais não convencionais como, por exemplo, telefones celulares, *tablets* etc. Esta miríade de dispositivos digitais recentes ou tradicionais, computacionalmente capazes, virtualmente conectados e eventualmente ociosos, se devidamente

coordenados, podem representar um potencial de processamento sem precedentes.

Um exemplo clássico de dispositivos com poder computacional relevante são os receptores de TV Digital [Morris e Chaigneau 2005], cuja presença nas residências é uma tendência com a digitalização da televisão, a mais popular das mídias de massa. A TV Digital oferece recursos que vão desde a melhoria da qualidade da imagem à capacidade de interação com o conteúdo. Com essa nova modalidade de TV, o telespectador tem a possibilidade de exercer um papel mais ativo, interagindo com os programas de televisão, que além de áudio e vídeo, passam também a incorporar *software* de forma sincronizada. Para tanto, o receptor de TV Digital conta com características típicas de um computador: possui memória, processador, sistema operacional e capacidade de se conectar em rede.

O grande alcance que a mídia televisiva apresenta com audiências que podem atingir bilhões de pessoas [BOB 2008], a exemplo de transmissões de eventos globais como olimpíadas e copas do mundo, demonstra bem a escala associada com este segmento. Na Europa, onde a TV Digital aberta já se encontra disponível, quatro milhões de receptores foram vendidos na Itália entre 2005 e 2007 [Freeman e Lessiter 2003]. A tendência é global e no Brasil em 2005 foi oficialmente iniciado o desenvolvimento do padrão brasileiro de TV Digital aberta, através do projeto SBTVD (Sistema Brasileiro de TV Digital) [Eduardo, Leite e Rodrigues 2005]. A partir de dezembro de 2007, o SBTVD entrou em um processo de implantação paulatina e já se encontra em operação na maioria das capitais e em diversas cidades.

Para demonstrar a viabilidade de implantação da arquitetura OddCI usando recursos não convencionais voláteis e distribuídos, nós modelamos um caso especial da arquitetura baseado na tecnologia usada em redes de TV Digital. Nós chamamos esta implementaçao de OddCI-DTV [Costa et al. 2009].

A organização do restante do capítulo é a seguinte: a Seção 6.1 traz uma revisão dos principais aspectos do segmento de TV Digital; a Seção 6.2 descreve como um sistema OddCI pode ser modelado sobre uma rede de TV Digital e a Seção 6.3 descreve como o protótipo OddCI-DTV foi desenvolvido e apresenta uma avaliação do seu desempenho baseado em um testbed real. Esta seção também traz uma análise dos resultados obtidos pelos dispositivos computacionais não convencionais quando comparados a alternativas mais tradicionais e, na Seção 6.4, fazemos as nossas considerações finais.

6.1 TV Digital Interativa

Uma importante convergência tecnológica está acontecendo em todo o mundo com a adoção crescente de Televisão Digital Interativa (TVDI). Entre outras melhorias, um sistema de TV Digital permite que o espectador desempenhe um papel ativo, uma vez que traz recursos para interatividade, fornecendo além de alta qualidade de vídeo e áudio também a possibilidade de execução de aplicações no receptor de TV.

Um sistema de TVDI pode ser entendido como um conjunto de definições que tornam possível a construção de dispositivos para transmissão e recepção de TV digital dentro de uma rede de TV digital. Com base em tais definições, uma estação de TV transmite para os receptores, por meio de uma rede de transmissão, os sinais de áudio e vídeo digitalmente codificados usando um padrão pré-definido de modulação. Junto com os sinais de áudio e vídeo codificados, outras informações podem ser enviadas para serem processadas pelos receptores, incluindo aplicações interativas. O receptor de TV digital é o dispositivo responsável por decodificar o sinal recebido, processar as informações adicionais agregadas e executar as aplicações recebidas juntamente com o áudio e vídeo. Usualmente, uma rede de TVDI também inclui um canal de interação que permite que os espectadores possam enviar informações de volta para a estação de TV. Uma representação gráfica de uma rede de TV Digital pode ser vista na Figura 6.1.

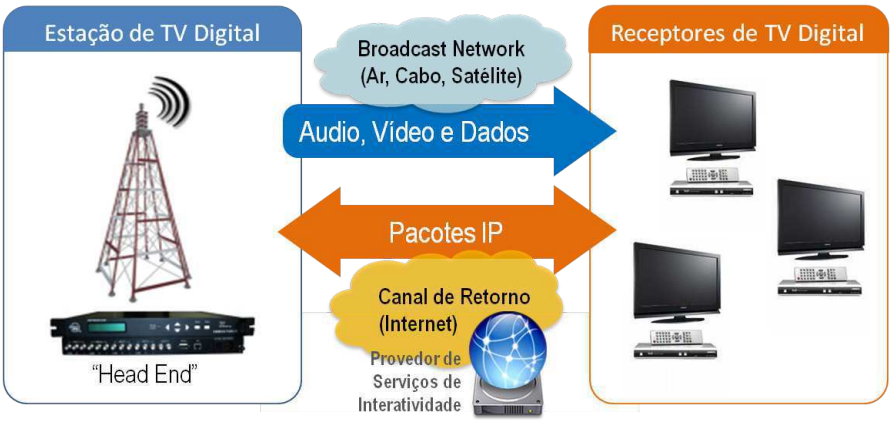


Figura 6.1: Estrutura padrão de uma rede de TV Digital

Na Europa, a TVDI já é um realidade com vários sistemas (*Digital Video Broadcasting* - DVB) [DVB 2011] em operação e milhões de dispositivos recebendo sinais digitais de TV [Freeman e Lessiter 2003]. Em muitos outros países, diversas iniciativas de implantação

de TVDI estão em andamento. No Brasil, o governo financiou a pesquisa que levou ao desenvolvimento do Sistema Brasileiro de TV digital (SBTVD) [Eduardo, Leite e Rodrigues 2005; Filho, Leite e Batista 2007]. Com o sistema já operando em várias regiões, espera-se uma adesão de até 80 milhões de usuários nos próximos anos [AB 2006]. Atualmente, existem em todo o mundo dezenas de milhões de receptores para processamento de sinal de TV digital já em operação e a tendência é uma ampliação desse contingente em um futuro próximo.

Os canais de transmissão de televisão digital terrestre podem atingir taxas de até 50 Mbps (DVB-T2) dependendo do sistema. No ISDB-T, utilizado no Brasil os canais possuem capaciade para transmissão de 19 Mbps. Em sistemas digitais com transmissão via satélite, como o ISDB-S, a taxa de um canal atinge 52 Mbps [Peng 2002]. No entanto, a transmissão de um vídeo de alta definição codificado com base no padrão MPEG-2 [ISO/IEC 1994] requer uma taxa de transmissão entre 10 e 18 Mbps [Fox 2002], e padrões mais recentes como o ITU H.264 [Wiegand et al. 2003] usam taxas menores ainda. Isso permite que algumas emissoras tenham mais de 30% da sua largura de banda de transmissão disponível para multiplexação de dados com o vídeo. Este excesso de capacidade pode ser usada para transmitir múltiplas legendas, múltiplos canais de áudio e vídeo, informações adicionais sobre os programas e também aplicativos para serem executados nos receptores.

O receptor de TV Digital (ou STB, do inglês *set-top-box*) pode ser visto como um computador adaptado para as necessidades do ambiente de televisão, tendo diversos processadores - um deles dedicado a executar aplicações interativas, memória, dispositivo de armazenamento não volátil, placa de rede, sistema operacional etc. Ele também executa um *middleware*, que é responsável por abstrair características de *hardware* específicas de cada receptor, permitindo que a mesma aplicação possa ser executada em *set-top-boxes* produzidos por diferentes fabricantes.

A maior parte dos *middlewares* disponíveis atualmente, tais como o DVB-MHP [DVB 2011; Morris e Chaigneau 2005] (*Digital Video Broadcasting - Multimedia Home Platform*) do padrão europeu, ATSC-ACAP [Morris e Chaigneau 2005] (*Advanced Television Systems Committee - Advanced Common Application Platform*) do padrão americano e o Ginga [Filho, Leite e Batista 2007] do padrão brasileiro suportam a linguagem Java como parte da solução para a execução de aplicações nos receptores. As

aplicações Java executadas nos receptores são chamadas *Xlets* [Batista C. E. C. F. 2006; Microsystems 2011].

Para permitir a execução de aplicações MHP em outras plataformas de TV digital, o DVB propôs o desenvolvimento de uma especificação unificada para *middlewares* de TV digital, chamada GEM (*Globally Executable MHP*) [ETSI 2004], incluindo características MHP que não estavam ligados a características específicas de receptores DVB. Esta especificação é atualmente adotada pelo padrões dos EUA e Japão (ATSC ACAP [Morris e Chaigneau 2005] e ARIB B.23 [ARIB 2004], respectivamente).

Também é importante notar que nem todos os programas de TV usam os recursos de interatividade do sistema TVDI e, quando usam, não necessariamente consomem todos os recursos disponíveis, gerando uma sobra de largura de banda no canal de transmissão e de capacidade de processamento do processador dedicado a aplicações interativas. Na verdade, devido à natureza da maioria dos programas transmitidos, é muito provável que esses recursos dificilmente sejam utilizados em 100% de sua capacidade o tempo todo.

Uma estação de TV digital abrange os elementos discutidos a seguir:

- **Codificador de Vídeo** (*Video Encoder*): É responsável pela codificação de um sinal de vídeo analógico em um fluxo de vídeo digital seguindo um determinado padrão (MPEG 2 ou H.264, por exemplo).

- **Gerador de Carrossel** (*Carousel Generator*): Em um sistema de TV digital, os dados e aplicações a serem transmitidos junto com o vídeo digital são normalmente codificados seguindo a especificação DSM-CC (*Digital Storage Media Command and Control*) [ISO/IEC 1998]. O DSM-CC suporta a transmissão de um sistema de arquivos utilizando o mecanismo de carrossel de objetos, que permite que grandes volumes de dados sejam transmitidos para um conjunto de receptores, repetindo ciclicamente a transmissão de seu conteúdo em unidades modulares. Os dados são repetidos ciclicamente para permitir que os receptores que sejam ligados no meio da transmissão ou aqueles que têm capacidade de processamento ligeiramente diferente dos demais possam ter acesso aos dados em momentos diferentes. Se um aplicativo no receptor deseja acessar um determinado arquivo do carrossel que já foi transmitido momentos antes, o acesso é adiado para a próxima retransmissão dos dados desse arquivo es-

pecífico. É possível atualizar dinamicamente o carrossel que está sendo transmitido, adicionando, removendo ou alterando os seus arquivos, através da criação de uma nova versão do módulo contendo os arquivos a serem atualizados. O *Carousel Generator* é responsável pela formatação do carrossel que precisa ser transmitido em cada momento específico.

- **Servidor de SI** (*Service Information Server*): Este componente é responsável pela gestão do banco de dados que contém as informações sobre os serviços oferecidos pela estação de TV (normalmente a programação de áudio e vídeo que a estação de TV transmite).

- **Multiplexador** (*Multiplexer*): Este componente é responsável pelo encapsulamento de todos os fluxos elementares (vídeo, áudio e dados) que precisam ser transmitidos juntos. A maioria dos sistemas adota o padrão ISO/IEC 13818 (MPEG-2) [ISO/IEC 1994].

- **Modulador** (*Modulator*: O objetivo do modulador digital é codificar um fluxo digital de bits para ser transferido através de um canal analógico. A técnica de modulação mais comumente usada em TV Digital é QAM (*Quadrature Amplitude Modulation*).

- **Transmissor** (*Transmitter*): Um transmissor é um dispositivo eletrônico que, com a ajuda de uma antena, propaga um sinal eletromagnético, tais como o usado em transmissões de rádio ou televisão. O sinal é então recebido e interpretado por um receptor.

A Figura 6.2 dá uma visão mais detalhada dos componentes internos de uma estação de TV de um sistema de TV Digital.

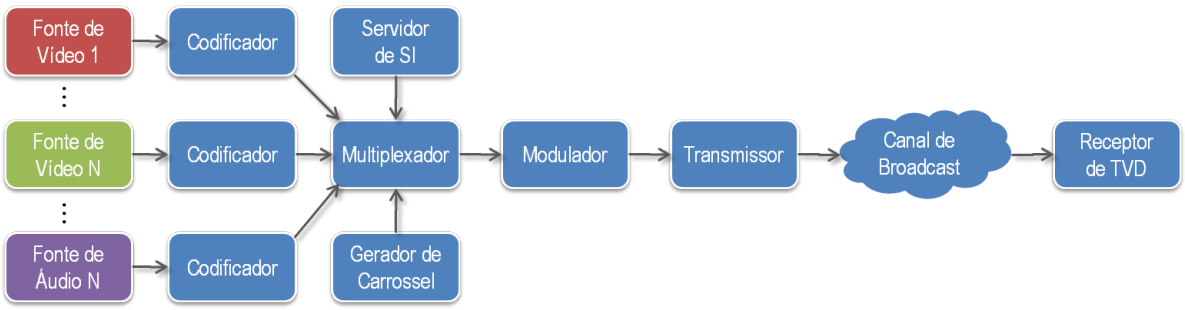


Figura 6.2: Arquitetura de um estação de TV operando um sistema digital

6.1.1 Executando Aplicações em um Receptor Interativo de TV Digital

Vamos agora descrever em mais detalhes como os aplicativos são transmitidos e executados no receptor de um sistema de TVDI. Como explicado anteriormente, a transmissão de dados da emissora para um receptor é realizada usando carrosséis de dados DSM-CC. Um carrossel de dados consiste em uma série de módulos, onde cada módulo pode, por sua vez, ser dividido em blocos para facilitar a transmissão. Carrosséis de objetos são construídos em cima do modelo de carrossel de dados. Eles estendem o carrossel de dados para adicionar o conceito de arquivos, diretórios e fluxos (*streams*). Isso permite que o carrossel possa conter um conjunto de diretórios e arquivos organizados em um sistema de arquivos tradicional.

Utilizando a abstração de um sistema de arquivos fornecido pelo carrossel de objetos, as aplicações e seus dados são continuamente transmitidos, multiplexados com áudio e vídeo e informações adicionais de controle (metadados). Esta informação é separada (*demultiplexed*) no receptor e adequadamente tratada pelo *middleware* e outros componentes.

Para sinalizar a um receptor que aplicações estão disponíveis, padrões de TVDI como o DVB e SBTVD definem uma tabela de informações de serviço chamada *Application Information Table* (AIT) [Morris e Chaigneau 2005; ETSI 2004; Eduardo, Leite e Rodrigues 2005]. A AIT contém todas as informações que o receptor precisa para executar a aplicação, como o nome, o identificador e o controle do ciclo de vida da aplicação. Este último é sinalizado pelo campo da AIT *application_control_code*, que permite que a emissora sinalize ao receptor o que fazer com a aplicação com relação à sua inicialização.

Aplicações com código de controle setado para AUTOSTART, também chamadas *trigger applications*, são carregadas e iniciadas automaticamente, sempre que o receptor está sintonizado em um canal de TV que está transmitindo essa aplicação. Assim, quando uma *trigger application* é transmitida no carrossel, ela é carregada por cada receptor que está (ou estará) sintonizado no canal associado. Um *trigger application* executará até o seu término ou até que outra *trigger application* seja transmitida no carrossel para o mesmo canal. Quando o receptor é desligado ou muda de canal, a execução da aplicação é interrompida.

Em um receptor de TV Digital, várias aplicações podem estar executando ao mesmo tempo e há, portanto, uma necessidade de impor uma separação entre as aplicações. Os *Xlets* são um conceito similar ao de *Applets* [Arnold e Gosling 1996]. Eles foram introduzidos pela Sun na especificação JavaTV e adotados como o formato de aplicação Java para o padrão

MHP e outros padrões relacionados com DTV. Como os *Applets*, a interface *Xlet* permite que um agente externo (o gerenciador de aplicações ou *Application Manager*, no caso de um receptor de TV digital) possa iniciar e parar uma aplicação, bem como controlá-la de outras maneiras.

Uma *Xlet* [Morris e Chaigneau 2005; ITVW 2011] deve estar, em todo o seu ciclo de vida, em um dos seguintes estados[1]: **Loaded**, **Paused**, **Started** e **Destroyed**. O diagrama de transição é mostrado na Figura 6.3:

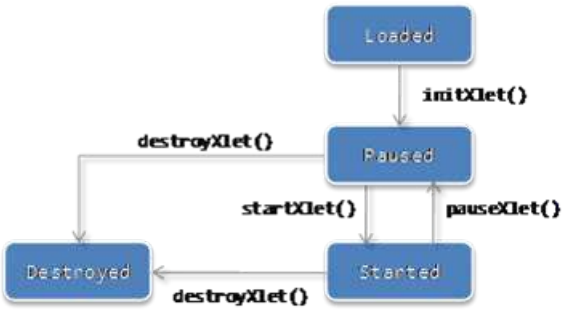


Figura 6.3: Diagrama de Estados de uma *Xlet*

O gerenciador de aplicações do *middleware* carrega a classe *main* do *Xlet* (conforme assinalada pela emissora) e cria uma instância da aplicação chamando o construtor *default*. Isto pode acontecer em qualquer momento após a aplicação ser recebida pelo receptor. Uma vez carregado, o *Xlet* fica no estado **Loaded**. Quando o usuário decide iniciar a aplicação (ou quando a emissora indica que o *Xlet* deve iniciar automaticamente - recurso usado no caso do PNA), o gerenciador de aplicações chama o método *initXlet*(), passando um novo objeto *XletContext* para o *Xlet*. O *Xlet* pode usar este *XletContext* para se inicializar e para carregar previamente qualquer recurso grande, como imagens, que demandem tempo para serem obtidas do carrosel de objetos que é continuamente transmitido pelo canal de *broadcast*. Quando a inicialização é finalizada, o *Xlet* fica no estado **Paused** e está pronto para iniciar a sua execução. Após receber o retorno do método *initXlet*, o gerenciador de aplicações do *middleware* chama o método *startXlet*(). Isto move o *Xlet* do estado **Paused** para o estado **Started** e o *Xlet* estará apto para interagir com o usuário, se for programada para fazer isto.

Durante a execução do *Xlet*, o gerenciador de aplicações pode chamar o método *pauseXlet*(). Isto faz com a aplicação seja movida de volta do estado **Started** para o estado **Paused**.

[1] A interface *Xlet* está disponível no pacote Java *javax.tv.xlet*.

A aplicação voltará para o estado **Started** novamente quando o gerenciador invocar novamente o método *startXlet*(). Isto pode acontecer várias vezes durante o ciclo de vida do *Xlet*. No final da execução do *Xlet*, o gerenciador de aplicações irá chamar o método destroyXlet(), o que levará o *Xlet* para o estado **Destroyed** e implicará na liberação de todos os recursos que foram alocados pela aplicação. Após este ponto, esta instância do *Xlet* não pode mais ser iniciada novamente [ITVW 2011].

6.2 OddCI-DTV: Um Sistema OddCI sobre uma Rede de TV Digital

Atualmente, diversas tecnologias já podem ser utilizadas para tornar possível a comunicação simultânea e unidireccional entre dispositivos digitais no modelo de um-para-muitos, característica do conceito de rede de *broadcast* evocado neste trabalho. Além da tradicional difusão de TV, em sua nova versão digital e em suas diferentes modalidades (satélite, terrestre, cabo, móvel etc) [Morris e Chaigneau 2005], também podemos citar a transmissão multicast por redes de banda larga, BitTorrent, redes de telefonia móvel e transmissão de vídeo (VoD, WebTV, IPTV etc). Ao tirar vantagem das funcionalidades já disponibilizadas em dispositivos que implementam tais tecnologias, ou complementando e/ou adaptando estas funcionalidades, é possível construir implementações de OddCI para vários contextos.

Da mesma forma, também é bastante ampla a diversidade de dispositivos que podem ser alcançados através de uma ou mais das tecnologias de transmissão mencionadas, de computadores a equipamentos com propósitos mais específicos, tais como consoles de jogos, telefones celulares e receptores de TV digital. Alguns destes dispositivos menos tradicionais já provaram o seu potencial de utilização para processamento distribuído em projetos de computação voluntária [Stanford 2011; Boincoid 2011].

Para demonstrar a viabilidade da arquitetura OddCI, nós construímos um protótipo baseado na tecnologia correntemente usada em redes de TV Digital (DTV). Nós chamamos esta implementação de OddCI-DTV e a Fig. 6.4 traz uma visão geral do seu funcionamento, o qual é aderente ao fluxo geral OddCI descrito na Seção 5.2.1.

6.3 Protótipo OddCI-DTV

Para instanciar a arquitetura OddCI sobre uma rede de televisão digital, é necessário implementar os três componentes de *software* que formam o núcleo de um sistema OddCI, ou seja: o *Provider*, o *Controller* e o PNA.

O papel do *Provider* pode ser exercido por uma rede de TV que produz e transmite programação nacional para diversas emissoras afiliadas. O papel do *Controller* pode ser exercido pela emissora/repetidora local de TVDI, a qual detém a concessão do canal de TV e será quem enviará, junto com sua programação, as mensagens de controle (dados) para os receptores conectados na sua frequência através de um fluxo elementar. Cada PNA é uma aplicação que executa sobre o *middleware* do receptor de TVDI, o qual no caso do SBTVD é chamado Ginga [Filho, Leite e Batista 2007]. O PNA usará a pilha TCP/IP e o canal de retorno (Internet doméstica), usado normalmente para interatividade, como um canal direto de comunicação com o *Controller* e o *Backend*.

A retaguarda (*Backend*), por sua vez, pode ser montada como um conjunto de servidores sob controle do *Client* ou de um terceiro, possivelmente usando recursos de um provedor público de computação na nuvem.

Na Fig. 6.5, são identificadas as tecnologias atualmente disponíveis para o segmento de TV Digital que podem ser usadas e como elas estão associadas com os elementos da arquitetura OddCI genérica.

Com base em tal mapeamento direto para os mecanismos nativos de TVDI, o modelo geral de operação de um sistema OddCI-DTV não requer muitas adaptações para o funcionamento sobre redes de TV Digital. Neste trabalho, nós assumimos um sistema de TVDI que é aderente ao padrão do Sistema Brasileiro de TV Digital (SBTVD).

Inicialmente, o *Client* solicita ao *Provider* a criação de uma instância OddCI, fornecendo a imagem da aplicação em um formato que permita que a mesma seja executada nos receptores de TV Digital. O *Provider* valida o *Client* e a imagem da aplicação e, baseado no histórico de audiência e em estimativas dos receptores conectados no momento, acata (ou não) o pedido.

Em seguida, o *Controler* formata e encaminha uma mensagem de controle para ser transmitida pela emissora de TV, incluindo na mesma uma versão de PNA compatível com os

receptores de TV Digital com o *flag* AUTOSTART setado. A emissora, após validar o *Controller* e a mensagem de controle, usa o seu transmissor para enviá-la. Para isso, é usado o processo de distribuição e execução de aplicações interativas, conforme descrito no padrão do SBTVD e que ocorre da seguinte forma: inicialmente o conteúdo da imagem da aplicação é serializado na forma de um carrossel de objetos no padrão DSM-CC [ISO/IEC 1998], onde os arquivos e pastas da aplicação são codificados em sessões e encapsulados em um fluxo *MPEG2 Transport Stream* (MPEG2-TS) [ISO/IEC 1994]. Após a codificação dos dados, as propriedades da aplicação como nome, tipo, classe principal e outras características são definidas e estruturadas através da tabela AIT (*Application Information Table*) e encapsulados em pacotes TS. Terminada a preparação dos dados, ocorre a configuração da tabela PMT (*Program Map Table*) com o PID utilizado pelo TS de dados (*Object Carousel*) e o PID da AIT, além da adição dos descritores necessários para identificar a existência de um fluxo de dados para um determinado programa ou serviço. Por fim, o fluxo de dados é multiplexado com outros fluxos de áudio, vídeo e dados. O fluxo combinado é então transmitido em *broadcast* pela emissora.

Todos os receptores de TVDI sintonizados no canal da emissora irão receber a mensagem de controle, representada por uma aplicação com o *flag* AUTOSTART ligado. Cada receptor verifica a existência do *stream* de dados, e executa uma rotina de processamento desses dados, a qual é responsável por verificar a integridade do conteúdo recebido através do CRC de cada informação. Os dados são gravados obedecendo à estrutura de pastas e arquivos configurados na AIT. Ao término do processamento, o *middleware* é notificado da existência de uma nova aplicação passando informações sobre o nome, o tipo e o modo de execução da aplicação para o gerenciador de aplicações que seleciona o módulo de apresentação (*engine*) adequado ao tipo de aplicação: NCL/Lua [ABNT 2009b] ou Java DTV [ABNT 2009c], por exemplo.

No nosso caso, a aplicação inicializada automaticamente é o PNA, que toma o controle e segue o fluxo OddCI normal (Fig. 5.3), usando o canal de retorno do receptor para sinalizar ao *Controller* a sua disponibilidade para participar da instância e, caso seja aceito, carregando a aplicação do cliente propriamente dita. A partir deste ponto, a própria aplicação do cliente usa o canal de retorno para obter tarefas e enviar resultados para o *Backend* diretamente.

6.3.1 O Componente PNA - *Processing Node Agent*

Como o *Processing Node Agent* (PNA) é o componente da arquitetura OddCI que executa nos dispositivos finais (nós de processamento), o mesmo precisou ser adaptado aos modelos de programação do *middleware* Ginga (Java e NCL) de forma a ser devidamente executado pelos receptores de TV Digital.

Conforme discutido na Seção 5.2.1, um PNA ativo possui dois estados: *Idle* e *Busy*. No estado *Idle*, o PNA não está integrando nenhuma instância OddCI mas fica monitorando o canal de *broadcast* permanentemente para o caso do *Controller* ter enviado alguma mensagem de controle do tipo WAKEUP convocando-o para integrar uma instância nova ou para recompor uma instância em andamento. Neste momento, o PNA passa do estado *Idle* para o estado *Busy*, carrega e executa a imagem da aplicação recebida e guarda a identificação (*id*) da instância que passou a integrar. Ele ficará neste estado até que um dos seguintes eventos ocorra; a) a aplicação finalize a sua execução ou b) receba uma mensagem do tipo RESET do *Controller* com a identificação da sua instância. Neste momento, o PNA libera os recursos usados pela aplicação e retorna para o estado *Idle*, reiniciando o ciclo. Em ambos os estados, o PNA periodicamente se comunica com o *Controller* através de sondas (*heartbeat messages*) contendo o seu estado e a identificação da instância à qual pertence, se estiver alocado à alguma.

Um trecho de código da versão do PNA em Java DTV que contém o seu algoritmo principal é mostrado na Figura 6.6.

6.3.2 Os Componentes *Provider*, *Controller* e *Backend*

O *Controller* e o *Backend* também foram implementados de forma completa e plenamente funcional, com aderência aos eventos básicos descritos no diagrama de sequência da Seção 5.2.1. Isto permitiu uma simulação completa de toda a dinâmica do sistema OddCI, com a interação do *Controller* com o PNA através da troca de mensagens de controle para criação e desmonte de instâncias, incluindo o envio da imagem da aplicação.

Para a validação do *Backend*, foi criada uma a aplicação paralela, chamda **Primos** com dois módulos: o *módulo cliente*, desenvolvido como uma aplicação que executa no receptor de TV Digital, e um *módulo servidor*, que executa em um computador convencional, o

qual representa o papel do *Backend*. O objetivo do módulo cliente é processar as tarefas que recebe do módulo servidor, que são caracterizadas por dois números representando um intervalo numérico discreto. O módulo cliente deve calcular todos os números primos existentes no intervalo e devolver o resultado para o módulo servidor. Neste ponto, solicita uma nova tarefa e o ciclo reinicia.

A aplicação Primos tem dois comportamentos possíveis: a) como aplicação BoT, no qual o módulo servidor distribui tarefas (intervalos de números) para os módulos clientes; e b) como aplicação paramétrica, na qual o próprio módulo cliente seleciona o intervalo numérico a ser processado. Em ambos os casos, a carga de processamento do módulo cliente pode ser regulado pelo tamanho do intervalo numérico a ser processado.

O papel do *Provider* foi simplificado no protótipo OddCI-DTV, com a assumpção de apenas um cliente que pede sempre a mesma instância, e embutido no *Controller*, que automaticamente dispara o pedido de criação desta instância padrão sempre que é inicializado.

6.3.3 Avaliando o Desempenho do Protótipo OddCI-DTV

Com o objetivo de realizar um estudo preliminar do desempenho do protótipo OddCI-DTV em receptores reais de TV Digital, foi construído um ambiente de testes (*testbed*) funcional que permitiu que todos os fluxos de comunicação fossem contemplados, como o fluxo entre o o PNA e o *Controller* (via os canais *broadcast* e direto) e a troca de informações entre a aplicação paralela e o seu respectivo *Backend* (via canal direto).

As subseções seguintes detalham quais as métricas que foram utilizadas na avaliação de desempenho, os experimentos realizados e também a configuração do ambiente usado nos testes.

Métricas de Desempenho

Três características específicas de um Sistema OddCI-DTV foram consideradas para aferição da eficiência do sistema implementado: a) a velocidade do *Controller* para disparar comandos pelo canal de *broadcast*; b) a capacidade do canal de retorno para receber tarefas a serem processadas e transmitir os resultados obtidos; e, finalmente, c) o potencial dos receptores de TV Digital para o processamento de aplicações paralelas. Neste sentido, as seguintes

métricas[2] de desempenho foram observadas:

- **Tempo Médio de Preparação do PNA ($\overline{\Sigma}$)**, o qual mede a velocidade do OddCI-DTV para criar instâncias e considera o tempo envolvido na comunicação *Controller*-PNA-*Controller* para iniciar a execução da aplicação. Ele é calculado pela expressão:

$$\overline{\Sigma} = w + d + r + a$$

  onde w é o tempo de preparação e transmissão da WM (contendo a imagem executável do PNA) do *Controller* para o receptor usando o canal de *broadcast* (carrossel de dados), d é o tempo de processamento do carrossel de dados e carga da imagem do PNA no receptor, r é o tempo para envio da solicitação de ingresso na instância do PNA para o *Controller* e a é o tempo para a resposta do *Controller* para o PNA.

- **Tempo Médio de Processamento ($\overline{\Lambda}$)**, o qual mede o tempo médio de processamento de diversas tarefas de uma aplicação pelo receptor de TV Digital a partir do momento em que o PNA inicia o processamento de uma tarefa até o momento em que é finalizado o processamento da mesma.

Descrição dos Experimentos

O primeiro experimento teve como objetivo medir o tempo de preparação do PNA ($\overline{\Sigma}$) usando aplicações de diversos tamanhos. Neste sentido, foram formatadas oito *wakeup messages* com tamanhos de 100, 500, 1.000, 1.500, 2.500, 3.500 e 7.500 Kb.

Foram também realizados experimentos para medir o tempo médio de processamento ($\overline{\Lambda}$) dos receptores de TV Digital. Um experimento usou a aplicação **Primos** com intervalos limites de diversas magnitudes. Os tamanhos escolhidos foram iguais a 10^n, com n variando de 1 a 6. No caso da aplicação **Primos**, a métrica $\overline{\Lambda}$ foi calculada através da divisão do tamanho do intervalo limite pelo tempo total de processamento.

Embora a aplicação **Primos** represente um exemplo real (fatoração de números primos possui grande utilidade na ciência em geral) e seja especialmente adequada ao objetivo do experimento: estressar a capacidade do receptor, nós também realizamos testes com uma

[2]Embora tenha sido usada a média em ambas as métricas também foram calculadas as suas medianas, as quais se mostraram equivalentes às médias sem apresentar diferenças relevantes.

aplicação de bioinformática real. A aplicação selecionada para os testes foi a BLAST (*Basic Local Alignment Search Tool*) [Altschul et al. 1990], um algoritmo de bioinformática para a comparação de informações de sequências biológicas primárias, tais como as sequências de aminoácidos de proteínas diferentes ou os nucleotídeos de sequências de DNA. Uma busca do BLAST compara uma sequência de consulta com uma biblioteca ou banco de dados de sequências, e identifica as sequências da biblioteca que se assemelham com a sequência de consulta, considerando um determinado limiar de similaridade fornecido. O código fonte do BLAST está disponível para *download* no sítio do *U.S. National Center for Biotechnology Information* (NCBI) [NCBI 2011]. Para os nossos experimentos, a versão da aplicação implementada em $C++$ foi portada usando um compilador cruzado (*cross compiler*) como uma aplicação residente do receptor de TV Digital - a qual executa diretamente no sistema operacional do mesmo. Para efeitos de comparação, as aplicações BLAST e **Primos** também foram executadas em um computador pessoal de referência.

Nós também conduzimos uma avaliação mais ampla da capacidade dos receptores de TV Digital considerando, além do PC de referência, recursos disponibilizados por provedores públicos de computação na nuvem. Para essa finalidade, nós realizamos uma análise cruzada usando os resultados de um *benchmarking* conduzido pela empresa Neustar/Webmetrics [Neustar 2011]. Os programas usados no *benchmark* foram portados para os receptores de TV Digital disponíveis e o seu desempenho pode ser avaliado usando a mesma referência. Novamente, os programas foram escritos em $C++$ e executaram como aplicações residentes.

Um último experimento envolveu uma aplicação que usa a pilha TCP/IP para buscar dados pelo canal de retorno. Foram realizados testes de acesso a páginas Web com 100, 500, 1.000, 1.500, 2.500, 3.500, 5.000, e 7.000 Kb usando um acesso doméstico padrão de 1 Mbps.

Exceto onde explicitamente definido de outra forma, todos os experimento foram replicados tantas vezes quanto necessárias para obtção de médias com intervalos de confiança de 95%.

Configuração do Ambiente de Testes

O ambiente montado para os testes envolve um sistema completo de transmissão e recepção de TV Digital (padrão SBTVD [ABNT 2009a]) disponível no Laboratório de Aplicações de Vídeo Digital da Universidade Federal da Paraíba (LAVID/UFPB), consistindo de: gerador de carrossel, multiplexador, modulador, transmissor (de baixa potência para uso local) e receptor TVDI de entrada (*low-end*) e topo de linha (*high-end*) com o *middleware* Ginga.

O *testbed* consiste dos seguintes componentes (sua configuração está detalhada na Tabela 6.1):

- Estação de TV para a formatação do carrossel de dados, multiplexação, modulação e transmissão das mensagens de controle para o *Controller*;
- Receptores de TV Digital para receber pelo ar e processar as mensagens de controle enviadas pela estação de TV;
- Duas versões do PNA (NCL/Lua e Java DTV), ambas implementando o comportamento descrito na Seção 5.2;
- Uma aplicação cliente em duas versões (Ginga-NCL/Lua and Ginga-J), a qual implementa o "Crivo de Eratosthenes" para encontrar números primos [TPG 2011];
- Duas aplicações residentes implementadas em C++: um algoritmo de bioinformática e um algoritmo para *benchmarking*;
- Versões do *Provider*, *Controller* e *Backend* desenvolvidos como serviços de rede e executados em PCs convencionais.

6.3.4 Verificação e Validação

Por se tratar de uma variação da arquitetura OddCI modelada sobre a tecnologia de TV Digital, a validação do modelo OddCI-DTV também não se aplica pelas mesmas razões citadas no capítulo anterior. Entretanto, nós realizamos algumas atividades de verificação para aferir se a especificação proposta para o protótipo foi devidamente obedecida na sua implementação. Usando testes de aceitação, análise de rastros e monitoramento da troca de

Tabela 6.1: Detalhes dos componentes do ambiente de testes do OddCI-DTV

Componente	**Descrição**
Estação de TV	Modulador Linear ISMOD (ISDB-T Digital Modulator - Série ISCHIO) e Gerador de Carrossel e Multiplexador Linear/DommXstream (Instalado em um servidor Intel(R) Xeon(R) x3430 2.4 GHz com placa Dektec, Memória RAM de 3 GB, Placa de Rede Gigabit Ethernet, S.O. Ubuntu Server 32 bits - v. 10.04); Taxa máxima do carrossel de dados configurada para 1Mbps.
Receptores de TV Digital	*Low-end*: Proview modelo XPS-1000 (firmware 1.6.70, middleware Ginga da RCA-Soft, com processador STMicroeletronics STi7001, Tri-core (audio, vídeo, dados) 266 MHz de clock, memória RAM de 256 MB DDR, memória flash de 32 MB, placa de rede Fast Ethernet (10/100) e Sistema Operacional adaptado do STLinux; *High-end*: PVR baseado no processador Intel CE 3100 com 1.06 GHz, RAM 256 MB DDR, Fast Ethernet (10/100) placa de rede Fast Ethernet e uma adaptação do sistema operacional Linux.
Processing Node Agent (PNA)	Versão A: em NCL/Lua Script [ABNT 2009b], imagem (executável) com $116,5Kb$. Versão B: em Java-DTV [ABNT 2009c], imagem de $20,3Kb$.
Aplicação Cliente	*Aplicação* ***Primos***, que implementa o algoritmo "crivo de Eratóstenes" para encontrar números primos até um valor limite. Implementada em duas versões: NCL/Lua e Java DTV, com tamanho do executável resultante em $2,6Kb$ e $10,8Kb$, respectivamente. *Aplicação de Bioinformática*: usando um compilador cruzado (*cross compiler*), foi-portado parte do *NCBI Toolkit* (programas *blastall* e textitblastcl3) para o receptor de baixo custo (*low-end*) usado. *Benchmarking da Bitcurrent*: Nós implementamos os mesmos algoritmos das tarefas de uso intensivo de CPU ($1.000.000$ de operações de seno e soma) e das tarefas de uso intensivo de entrada e saída (busca sequencial por um registro em um arquivo com 500.000 registros e com tamanho de $128MB$), conforme descritos na metodologia do *benchmarking* da Bitcurrent, para os dois tipos de receptores usados nos testes (*low-end* e *high-end*).
Provider, *Controller* e *Backend*	O *Provider*, *Controller* e *Backend* foram implementados como serviços de rede executando sobre o *middleware* Apache/Tomcatv6.0.33, protocolo HTTP para troca de mensagens, *scripts* do *framework* Web Grails/Groovy, MySQL v.5.1 para o armazenamento de tarefas e resultados no *Backend*. No caso do *Provider*, foi criada uma interface Web para que clientes solicitem a criação de instâncias e a comunicação com o carrossel de dados. Estes componentes foram executados em um computador com processador Intel(R) Xeon(R) x3363 2.83 GHz, Memória RAM de 512 MB, Placa de Rede Gigabit Ethernet e SO Ubuntu Server 32 bits v9.10.
Computador Pessoal de Referência	Para fins de comparação de desempenho com os receptores TVDI foi usado um *notebook* com Processador Intel(R) Core(TM) i3-2310M 2.1 GHz, Memória 4 GB RAM, Placa de Rede Fast Ethernet e SO Ubuntu 64 bits v11.10.

mensagens entre os diversos componentes do protótipo, foi feita uma verificação da dinâmica do funcionamento real com relação ao modelo proposto.

Algumas simplificações foram realizadas na especificação para facilitar a implementação. Dentre elas, não foi implementado um DVE real nas duas versões do PNA usadas, cuja criação foi apenas simulada pela ativação de um método vazio. A solicitação de instâncias entre o *Provider* e o *Controller* não envolveu a análise de viabilidade de atendimento da demanda. Todas as demandas eram automaticamente aceitas. O processo de coordenação não considerou a ativação de mecanismos compensatórios no *Controller*, apenas o envio de mensagens de controle para a criação de instâncias.

A verificação das três versões do algoritmo do ”crivo de Eratóstenes”que foram usadas (em Java, em NCL/Lua e em Java DTV) foi realizada através da comparação entre as saídas produzidas para diversos intervalos usados como parâmetros de entrada. O algoritmo foi portado com a máxima fidelidade em cada uma das linguagens para garantir que a mesma computação fosse realizada em cada ambiente e os resultados produzidos pelas três versões foram comparados para reforçar essa condição.

No caso do *toolkit* NCBI e dos programas *Blastall* e *Blastcl3* não houve alteração de código. O mesmo código original foi compilado tanto no PC de referência quanto no STB usado nos testes. As saídas produzidas nos treze testes realizados no PC de referência e no STB foram então comparadas e verificadas para garantir que os mesmos resultados e, consequentemente, o mesmo processamento foi realizado nos dois ambientes.

O mesmo ocorreu no caso da replicação do *benchmarking* da *Bitcurrent*. Os algoritmos dos dois testes que foram replicados, CPU e I/O, foram implementados uma única vez e compilados no PC de referência e nos dois tipos de STB utilizados. Novamente, os resultados produzidos nos três ambientes foram comparados e verificados.

6.3.5 Resultados e Análise

O resultado das medições dos tempos médios para preparação do PNA para vários tamanhos de imagens obtido no primeiro experimento está exibido na Figura 6.7. Esta análise mostra que o tempo de preparação pode ser estimado com segurança, desde que o mesmo depende, principalmente, do tamanho da imagem da aplicação e do tempo necessário para a sua transmissão em *broadcast* e há pouca dependência dos demais fatores envolvidos.

Para comparar a capacidade de processamento de um receptor com um computador pessoal de referência, o módulo cliente da aplicação **Primos** foi executado em ambas as plataformas. O resultado apresentado na Figura 6.8 (escala logarítmica) demonstra que o receptor *low-end* é, em média, 27 vezes mais lento do que o PC de referência. Outra observação é que a aplicação estoura a memória no receptor *low-end* quando tenta processar números acima de 10^6.

No caso da aplicação de bioinformática BLAST, os testes representaram diferentes cargas de trabalho e foram realizados usando os programas *blastall* e *blastcl3*. Um total de 15 experimentos foi executado no receptor *low-end* tanto no modo "em uso", com um canal de TV sintonizado, quanto no modo "standby", com o *middleware* em um estado inativo. Eles foram divididos em três categorias: processamento local da busca em bibliotecas de sequências com pequeno volume de registros (testes de 1 a 9), processamento local da busca em bibliotecas de sequências com grande volume de registros (testes de 10 a 12) e processamento remoto, feito contra as bibliotecas do próprio NCBI (testes de 13 a 15). Os mesmos testes foram reproduzidos no PC de referência. Os resultados obtidos para as primeiras duas categorias são mostrados na Tabela 6.2, enquanto que os resultados da última categoria são apresentados na Tabela 6.3.

O programa *Blastall* foi executado com diferentes parâmetros de entrada para apuração da redução de desempenho do receptor *low-end* com relação ao PC de referência. Para comparar o desempenho, calculamos as médias dos tempos de resposta da aplicação executando em cada ambiente com um intervalo de confiança de 90%, conforme apresentado na Tabela 6.2. O desempenho médio do receptor *low-end*, quando comparado com o PC de referência, foi $20,6$ vezes pior com um erro máximo de $\pm 10\%$. Os resultados também mostram que a redução média de desempenho quando se compara os tempos de execução do receptor no modo *standby* e em uso normal é $1,65$ vezes, com um erro máximo de $\pm 17\%$.

Os reultados dos testes para medir o desempenho do canal direto estão exibidos na Figura 6.9 (escala logarítmica). Através de um programa simples que usa o canal de interação do receptor para obter dados do *Backend*, testes foram realizados para acessar páginas Web com tamanhos com 100, 500, 1.000, 1.500, 2.500, 3.500, 5.000, e 7.000 Kb usando uma conexão doméstica padrão de $1Mbps$.

O computador de referência acessou as diferentes páginas sem maiores dificuldades, en-

Tabela 6.2: Tempos de processamento obtidos na execução do programa *Blastall* no receptor TVDI e no PC de referência (em segundos)

#Teste	Receptor TVDI		PC com Linux x86
	Em Uso	*Standby)*	
1	3,34	1,36	0,56
2	2,10	1,33	0,04
3	5,18	3,21	0,08
4	0,18	0,18	0,01
5	0,17	0,12	0,02
6	0,17	0,12	0,01
7	1,03	0,61	0,29
8	0,94	0,61	0,02
9	1,64	0,09	0,02
10	0,18	0,12	0,01
11	9.314,25	6.315,41	213,77
12	38.858,30	26.973,26	747,37

quanto que a aplicação executando no receptor *low-end* enfrentou problemas de memória com páginas a partir de $2.500Kb$. Assim, para comparação, foi calculado o tempo projetado para páginas acima de $2.500Kb$ no receptor TVDI, com uso de regressão linear. O tempo do receptor é, em média, 19 vezes maior do que o computador de referência com intervalo de confiança de 95%. A diferença é menor do que nos testes anteriores anterior porque envolve o tempo de tráfego dos dados no enlace, o qual tem impacto em ambos os ambientes.

Também foi verificada a capacidade do receptor *low-end* para se comunicar adequadamente com o *Backend* através do canal direto para a obtenção de tarefas e para enviar resultados usando o programa *blastcl3*. Este programa submete uma sequência para ser procurada nas bases de dados do NCBI, recebe o resultado e grava-o em um arquivo. Como o processamento de busca é executado remotamente, o aspecto mais relevante neste experimento é a maneira com que o STB manipula dados sobre as conexões de rede. Neste caso, como pode ser verificado na Tabela 6.3, não há diferença de desempenho significativa entre o PC de referência e o receptor *low-end*. Uma eventual sobrecarga nos servidores do NCBI ou tráfego de rede pode ser a causa do resultado do teste 13, no qual o receptor levou menos tempo do que o PC para completar a tarefa.

Tabela 6.3: Tempos de processamento obtidos na execução do programa *Blastcl3* no receptor TVDI e no PC de referência (em segundos)

#Teste	**Receptor TVDI**		**PC com Linux x86**
	Em Uso	***Standby***	
13	79,28	77,39	114,24
14	84,92	89,88	82,16
15	449,19	436,17	445,05

Nós também comparamos o desempenho de receptores TVDI com o desempenho de máquinas virtuais oferecidas por provedores públicos de computação em nuvem. Na comparação, foi usado o *benchmarking* conduzido pela equipe Bitcurrent [Bitcurrent 2011], Foram realizados os mesmos testes de processamento intensivo (CPU) e uso intensivo de dados (I/O) tanto nos receptores *low-end* quanto nos receptores *high-end*. Os resultados estão consolidados na Tabela 6.4 (média dos tempos em segundos com intervalo de confiança de 95%).

Tabela 6.4: Resultados do *Benchmarking* de CPU e IO dos Receptores TV Digital (em segundos)

Teste	**Receptor TV Digital**	
	ST 7109	**CE 3100**
Teste de CPU	2,55	0,19
Teste de IO	12,90	1,48

Os resultados completos da avaliação de desempenho realizada estão consolidados em um relatório [Neustar 2011]. A Tabela 6.5 apresenta um resumo desses resultados.

Tabela 6.5: Resultados do *Benchmarking* Bitcurrent (em segundos)

Teste	**Serviço Público PaaS/IaaS**				
	Salesforce	**Google**	**Rackspace**	**Amazon**	**Terremark**
GIF de 1x1 pixel	0,11	0,25	0,18	0,23	0,23
GIF de 2 MBytes	0,50	1,97	3,25	4,41	5,00
Teste de CPU	8,13	1,63	2,16	10,03	3,75
Teste de IO	6,26	2,03	3,33	19,46	12,35

Como pode ser visto, ambos os receptores de TV Digital obtiveram desempenho similar

ou superior aos obtidos pelas plataformas convencionais de IaaS e PaaS, especialmente para o teste de CPU. Embora os testes acima tenham sido realizados enquanto os dispositivos estavam ociosos, em modo "standby", nós também testamos os receptores de TV Digital durante sua operação normal (quando o usuário está assistindo TV). A perda de desempenho observado foi de 33% para o receptor *low-end* e de 15% para o receptor *high-end*, mas os resultados mantiveram-se próximos aos obtidos nos provedores de computação na nuvem. Ressaltamos que esta é uma comparação incompleta, porque não temos os intervalos de confiança do *benchmarking* da equipe Bitcurrent.

Este resultado pode ser explicado pelos processadores poderosos presentes nestes dispositivos e pelo fato de que eles estavam dedicados ao processamento dos testes.

A avaliação da capacidade de processamento do receptor *low-end* utilizado mostrou que ele é, em média, 27 vezes mais lento que um computador pessoal típico. Como os testes envolveram receptores de baixo custo, representando o pior caso, e a tendência observada é de melhoria da capacidade dos equipamentos, espera-se que esta relação possa ficar mais favorável, como pode ser visto no caso do receptor *high-end*. Entretanto, o fato do receptor ser mais lento não é necessariamente um problema, uma vez que a escala potencial de uma rede de TV Digital é da ordem de centenas de milhares ou milhões de vezes maior do que uma grade computacional tradicional, por exemplo.

As limitações de memória do receptor observadas durante os experimentos devem ser consideradas para definir o perfil adequado para as aplicações que irão executar em instâncias OddCI. Como a filosofia das aplicações BoT é que elas podem ser muito pequenas, é perfeitamente viável encontrar aplicações cujos requisitos principais são de processamento. Há casos em que o uso de memória é pequeno e constante (o qual não aumenta a alocação com o tempo), a exemplo de aplicações que buscam padrões. Desta forma, ajustes na granularidade das tarefas da aplicação BoT podem permitir o aproveitamento apropriado dessa infraestrutura.

Nos experimentos, foi possível verificar que o canal de *broadcast* da TV Digital mostrou-se eficiente para os propósitos do OddCI-DTV. Um canal SBTVD dispõe de uma banda total entre 18 e 21 Mbit/s, a depender de configuração [ABNT 2009b; ABNT 2009c]. A experiência mostra que emissoras podem dispor de uma banda residual de 1 a 4 Mbit/s para o carrossel de dados, considerando a vazão necessária para um fluxo de vídeo full HD codifi-

cado em H.264 e uma margem de segurança. Com 1 Mbit/s, o *wakeup process* inicial usando uma aplicação BoT típicos consome apenas algumas dezenas de segundos.

Avaliação da Segurança

No contexto da TV Digital, algumas soluções de segurança estão disponíveis em várias partes de sua arquitetura [Morris e Chaigneau 2005], como embaralhamento de sinal (*signal scrambling*), confidencialidade baseado em PKI e SSL/TLS no canal direto, a assinatura de aplicações, *sandbox*, *proxies* intermediários para os recursos do dispositivo e perfis de autorização de uso dos recursos disponíveis.

Uma validação preliminar dos conceitos de segurança de um sistema OddCI-DTV foi realizada tendo como base a especificação do *middleware* do SBTVD [ABNT 2009a], a qual define as linguagens que podem ser utilizadas para codificação das aplicações e as interfaces de programação (APIs, do inglês *Application Program Interface*) disponíveis.

Neste sentido, parte das primitivas de segurança descritas no modelo de segurança definido na Seção 5.3 foram implementadas nas linguagens NCL/Lua [ABNT 2009b] e Java DTV [ABNT 2009c] ou mapeadas para recursos nativos desses ambientes. Tomando por exemplo uma aplicação Java DTV, uma API de segurança complementar é especificada no pacote *com.sun.dtv.security* que estende a API *java.security*. De forma similar, para as aplicações implementadas em NCL/Lua é possível fazer uso da biblioteca aberta Lua MD5 [Kepler 2010], a qual já inclui uma implementação dos algoritmos $MD5$ e $des56$.

Conforme a normatização do SBTVD, o *middleware* que está instalado nos receptores deve fazer automaticamente a validação de cada aplicação recebida usando a chave pública da emissora que está assinada por uma autoridade certificadora bem conhecida. Além disso, os mecanismos nativos que estão previstos para proteger os recursos e o funcionamento do *middleware* contra o comportamento indevido de aplicações interativas, seja ele intencional ou não, podem ser mapeados para obter o mecanismo de DVE previsto. Estes ambientes disponibilizam uma quantidade controlada de recursos para a aplicação em execução, garantindo dessa forma, a manutenção dos principais serviços dos dispositivos hospedeiros e preservando a plataforma de possíveis ataques de alocação de recursos.

O esforço prévio para identificar e decompor as vulnerabilidades e mapeá-las em primitivas básicas (ver Seção 5.3) permite aplicar as mesmas técnicas que já foram validadas em

outros contextos. Com esta estratégia, foi possível relacionar as primitivas básicas com os recursos presentes em sistemas de TV Digital que atendam as normas estabelecidas. Todas as primitivas necessárias para a operação segura de um sistema OddCI ou já fazem parte das bibliotecas padrão de um sistema de TV digital, ou podem ser construídas de forma trivial usando estas bibliotecas. A implementação de algumas delas serviu para provar a viabilidade do seu desenvolvimento.

6.4 Considerações Finais

Nós discutimos como um sistema OddCI pode ser implementado sobre tecnologias atualmente disponíveis e apresentamos os resultados que alcançamos na modelagem da arquitetura OddCI sobre uma rede tradicional de TV Digital, que chamamos de OddCI-DTV.

A construção de uma prova de conceito com a implementação do sistema OddCI-Ginga sobre uma rede de TV Digital, a montagem de um *testbed* real e uma avaliação do seu desempenho mostraram não apenas a viabilidade dessa abordagem como também o fato de que ela pode representar um caminho promissor.

Em particular, esta fase da pesquisa permitiu obter medições de campo sobre o potencial da TVD para sistemas OddCI. Assim, foi possível confirmar o comportamento linear na transmissão de mensagens de controle por radiodifusão, a adequação dos recursos de comunicação direta dos receptores para troca de tarefas/resultados e algumas das eventuais limitações de processamento dos dispositivos.

Os testes em um ambiente real permitiram identificar também as limitações potenciais do receptor, notadamente com relação à memória. Isso deve ser usado para definir o perfil das aplicações adequadas para instâncias OddCI. Acreditamos que é perfeitamente viável encontrar aplicações cujos requisitos principais são de processamento. Há casos em que o uso de memória é pequeno e constante (que não aumenta a alocação com o tempo), a exemplo de aplicações de reconhecimento de padrões. Além disso, eventuais ajustes na granularidade das tarefas da aplicação BoT podem permitir um adequado aproveitamento dessa infraestrutura.

Por outro lado, a enorme quantidade de dispositivos não convencionais existentes e sua capacidade potencial combinada de processamento indicam que é possível montar estrutu-

ras OddCI poderosas e altamente elásticas para atender demandas específicas de aplicações HTC.

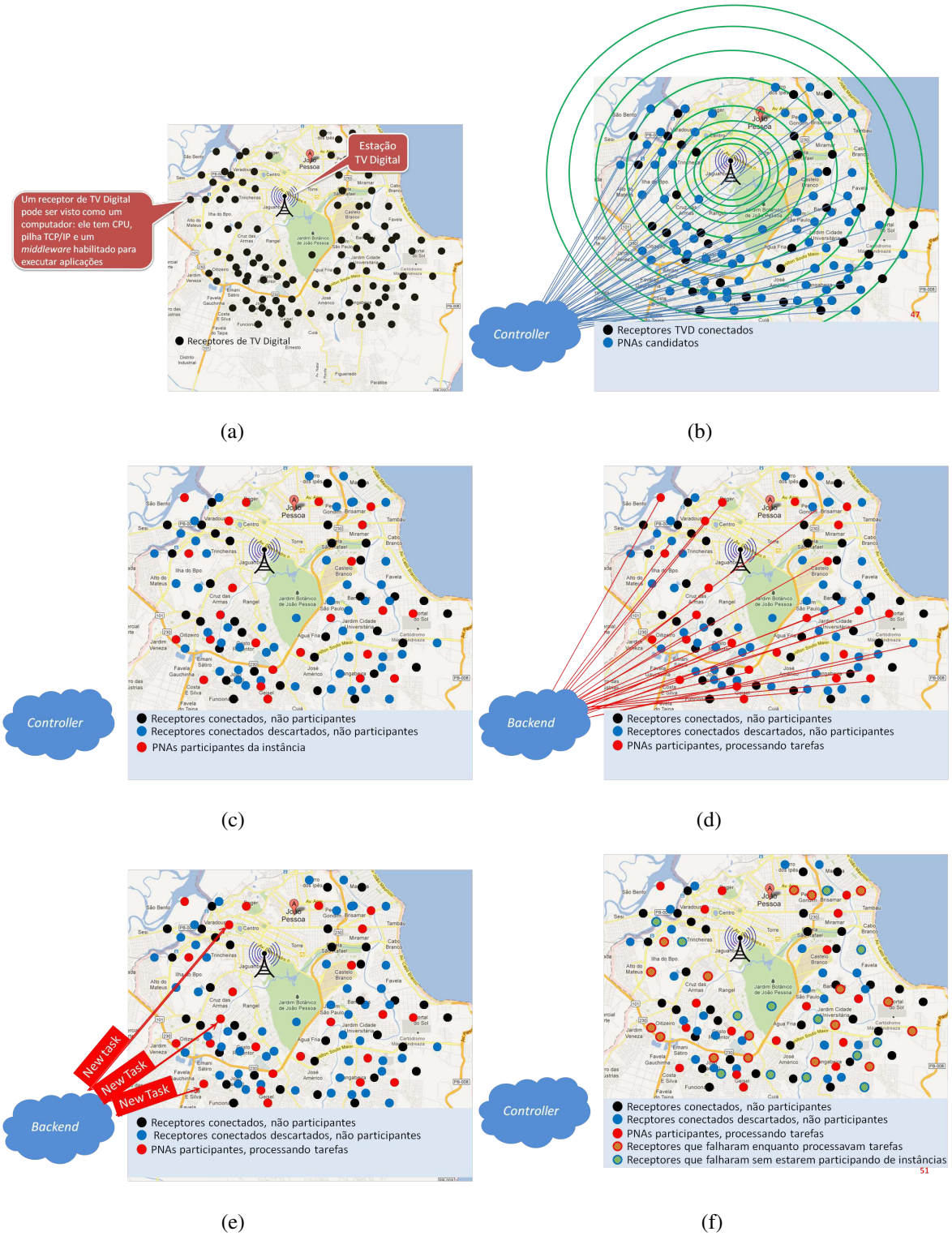


Figura 6.4: Visão Geral OddCI-DTV: Uma rede básica de TV Digital é composta por uma estação e por receptores (a); o *Controller* usa a estação para enviar WMs, as quais são respondidas por uma fração controlada dos dispositivos conectados (b); o *Controller* seleciona parte dos dispositivos respondentes e descarta os demais (c); os dispositivos aceitos para a instância contactam o *Backend* para obter tarefas (d) e devolver os resultados (e), repetindo o ciclo até o final do processamento; eventuais falhas precisam ser repostas pelo *Controller* através de novas WMs (f)

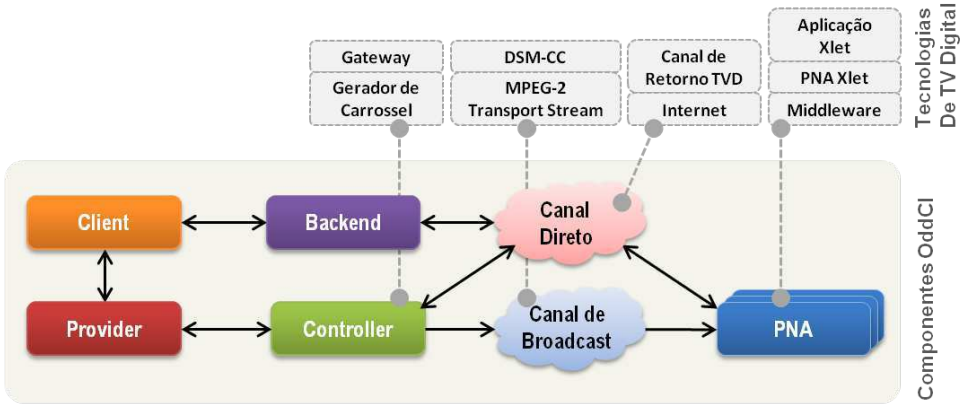


Figura 6.5: Mapeamento de um Sistema OddCI sobre tecnologias atuais de uma rede de TVDI

```
public class PNA {
    protected int state = PNAState.IDLE;
    protected String pna_id = "EMPTY";

    public void startXlet() throws XletStateChangeException  {
        while (true) {
            pna_id = control.sendHBI(pna_id, state);
            if (!pna_id.equals("EMPTY") && control.hasMessage()) {
                switch (state) {
                    case PNAState.IDLE:
                        if (control.get(PNAAttribute.MSGTYPE).equals("WAKEUP")) {
                            state = PNAState.BUSY;
                            vm = newVMThread(control.get(PNAAttribute.APPIMAGE));
                            vm.start();
                        }
                        break;
                    case PNAState.BUSY:
                        if (!vm.isAlive()) {
                            state = PNAState.IDLE;
                        } else {
                            if (control.get(PNAAttribute.MSGTYPE).equals("RESET")) {
                                if (vm.isAlive()) {
                                    vm.stopped = true;
                                }
                                if (finalize != null) {
                                    finalize.run();
                                }
                                state = PNAState.IDLE;  // the PNA is free again
                            }
                        }
                        break;
                }
            }
        }
    }
}
```

Figura 6.6: Algoritmo Principal do PNA em Java DTV

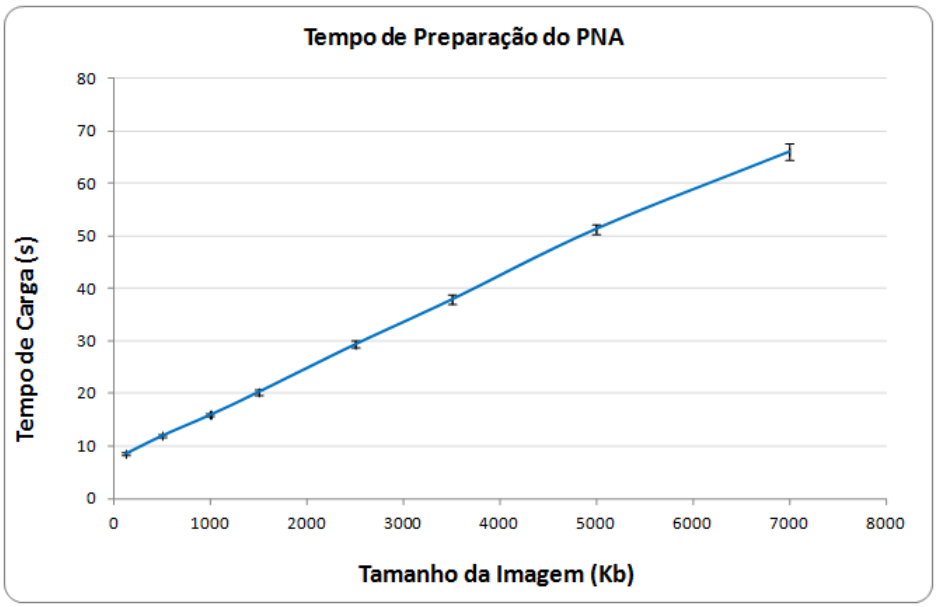


Figura 6.7: Tempo de carga do PNA

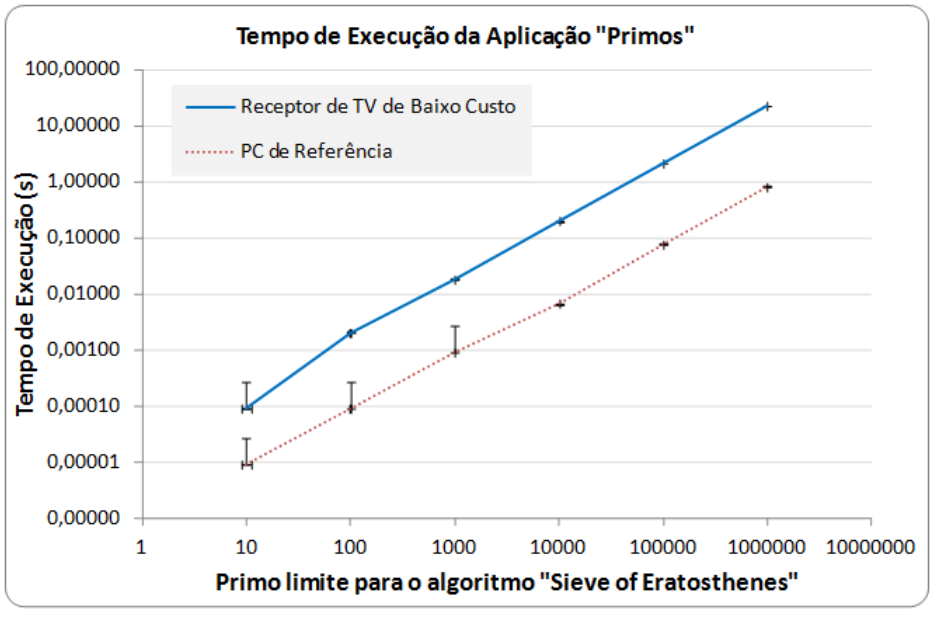


Figura 6.8: Comparação do tempo de execução da aplicação **Primos**

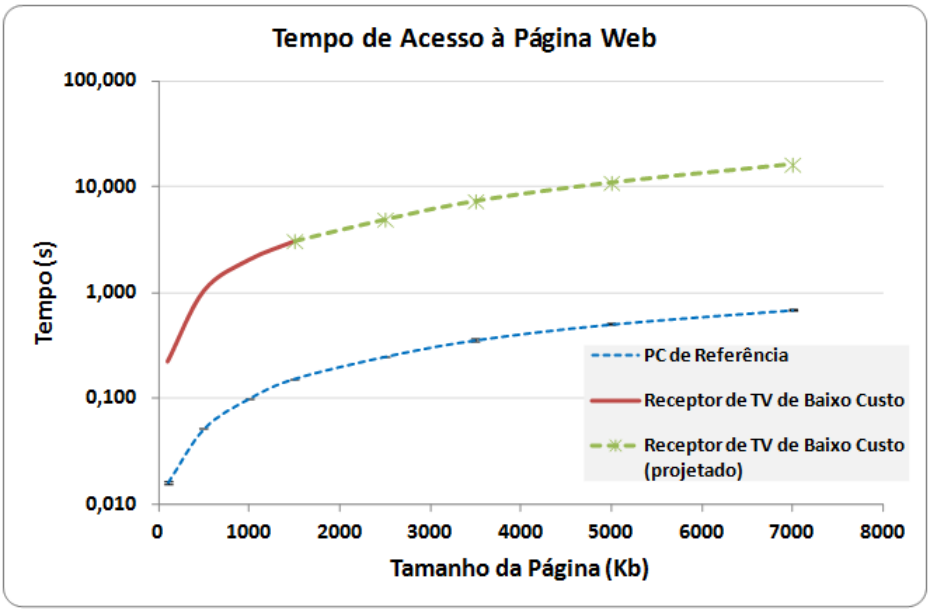


Figura 6.9: Comparação do tempo de acesso a uma página Web

Capítulo 7

Trabalhos Relacionados

7.1 Abordagens Alternativas para Provimento de Recursos

O *RESERVOIR Project* [Rochwerger et al. 2009] apresenta uma arquitetura que permite que os provedores de infraestrutura de nuvem possam compartilhar recursos de forma dinâmica uns com os outros para criar um *pool* virtualmente infinito de recursos. Seu modelo de computação na nuvem federada é baseado na separação entre os papéis funcionais de provedores de serviços e provedores de infraestrutura, onde os últimos podem arrendar recursos dinamica e transparentemente para os primeiros. A arquitetura OddCI pode ser aplicável para esse contexto.

A abordagem *InterClouds* [Buyya, Ranjan e Calheiros 2010] endereça o problema de provisionamento de nuvens usando uma federação orientada para o mercado de locação de recursos. Baseado na intermediação através de um mercado de câmbio, corretores de nuvens organizam a relação entre os consumidores de serviços e coordenadores de nuvem em ambientes de nuvem distribuídos. No entanto, as lacunas na integração e interoperabilidade entre os fornecedores de nuvem limitam a sua viabilidade.

Experiências como *Ad hoc cloud* [Kirby et al. 2010] que permitem a virtualização parcial de *hardware* não dedicado, e *Nebulas* [Chandra e Weissman 2009], baseado em recursos voluntários distribuídos, confirmam a possibilidade de utilizar recursos de uso geral com granularidade muito alta para a construção de *JiT Clouds*.

Os *Nano Data Centers* (*NaDa*) [Valancius et al. 2009] visam habilitar uma infraestrutura distribuída de borda para hospedagem e armazenamento de dados e distribuição de conteúdo.

Como também suportado pelas *JiT Clouds*, a abordagem *NaDa* é baseada em recursos não convencionais, mas com propósitos mais específicos. As duas principais aplicações planejadas para *Nano Data Centers* são vídeo sob demanda e jogos multiusuários.

O trabalho de Menascé e Ngo [Menascé e Ngo 2009] discute como os métodos tradicionais de planejamento de capacidade foram impactados com o advento da computação na nuvem e como os riscos e custos envolvidos estão migrando dos clientes para os provedores. O aprofundamento que fizemos nos aspectos de disponibilidade e regulação da demanda por parte dos provedores confirma esta condição.

Anandasivam *et al.* [Anandasivam, Buschek e Buyya 2009] introduzem uma versão do conceito de preço auto-ajustável adaptada para computação na nuvem, no qual o provedor usa um sistema de leilão que atua como uma influência no comportamento de usuários sensíveis ao preço e regula o uso dos recursos disponíveis. Nosso estudo mostra que o limite imposto pelos provedores também pode ser usado como um regulador da demanda dos usuários. De fato, uma observação da situação atual no mercado de IaaS mostra que esta é uma opção que é praticada por quase todos os fornecedores de IaaS.

7.2 Provisionamento e Coordenação de Recursos sob Demanda

Dentro do nosso conhecimento, nós somos o primeiro grupo a investigar o potencial do uso de redes de *broadcast* para a construção de infraestruturas computacionais distribuídas instantâneas e sob demanda [Batista et al. 2007] [Costa et al. 2009]. Existem, entretanto, alguns outros trabalhos que apresentam convergência com a nossa pesquisa.

O *framework* FALKON (*Fast and Light-weight tasK executiON*) [Raicu et al. 2007; Raicu et al. 2008] tem como foco a possibilidade de execução rápida de aplicações HTC em *clusters* computacionais baseando-se na integração de escalonadores multi-nível e despachantes (*dispatchers*) simplificados para oferecer alto desempenho. O escalonamento multi-nível do FALKON separa a aquisição de recursos (através de requisições em lote para escalonadores, por exemplo) da distribuição de tarefas, em um processo similar ao da abordagem OddCI.

O SNOWFLOCK [Lagar-Cavilla et al. 2009] é, por sua vez, uma implementação de uma

abstração de *fork* de máquina virtual que instantaneamente duplica uma VM em múltiplas réplicas executando em diferentes servidores através do uso de um esquema de comunicação um-para-muitos, como os sistemas OddCI. Usando uma técnica de distribuição multicast, SNOWFLOCK fornece uma eficiente clonagem em memória de VMs ativas que, potencialmente, pode escalar para centenas de réplicas consumindo poucos recursos de I/O da nuvem. Assim como o OddCI, SNOWFLOCK também aborda a instanciação, sob demanda, de milhares de VMs paralelas em determinados ambientes de computação na nuvem, mas que, diferentemente da nossa abordagem, requer a pré-alocação de recursos físicos e a integração de sua API nas aplicações em tempo de compilação.

Em termos de alocação de recursos sob demanda, o projeto NEPHELE [Warneke e Kao 2009] foi um dos primeiros *frameworks* para processamento paralelo que, explicitamente, buscou explorar a alocação dinâmica de recursos para escalonamento e execução de tarefas em ambientes de nuvem. Baseando-se em grafos de execução (*execution graphs*) elaborados pelo usuário, o *framework* NEPHELE também traz a possibilidade, como o OddCI, para alocar e desalocar, automaticamente, recursos computacionais durante a execução de uma aplicação.

Francois *et al.* [Francois, State e Festor 2007a] mostram que *hackers*, quando usando *botnets*, enfrentam os mesmos problemas de coordenação escalável endereçados no Capítulo 5. Uma *botnet* é uma rede de computadores comprometidos (*bots*) controlados remotamente por um *botmaster*. Estas estruturas provaram sua eficiência no controle de redes P2P com mais de 400.000 nós [McLaughlin 2004]. O uso de soluções de gerenciamento de serviços de rede inspirados em modelos de *malware* para controle de redes de larga escala foi proposto por Francois *et al.* em trabalhos subsequentes [Francois, State e Festor 2007b; Francois, State e Festor 2008]. Os principais benefícios destes modelos são: a) a capacidade de gerenciar um grande número de nós heterogêneos, e b) flexibilidade no uso, porque os controles e mecanismos de propagação são independentes das aplicações.

Desde que milhões de PNAs ativos podem estar enviando *heartbeat messages* para o *Controller*, simultaneamente, mecanismos de hierarquização, otimização e distribuição de frequência de envio devem ser incorporadas ao manuseio de tais mensagens para que as mesmas não representem um gargalo no sistema. Abordagens para problemas similares já foram propostas em outros contextos [Francois, State e Festor 2007a].

Na outra extremidade do processo, a infraestrutura de retaguarda precisa estar devidamente aprovisionada para usufruir plenamente da potencial vazão de processamento suportada pela instância OddCI criada. Neste sentido, a taxa na qual o *Backend* consegue despachar tarefas para os dispositivos pode limitar o poder de computação potencialmente disponível na instância OddCI. Entretanto, há diversas abordagens que podem ser adotadas na montagem do *Backend* para impedir que o mesmo seja um gargalo para o sistema. Um exemplo de abordagem aplicável é o projeto do servidor de tarefas (*Task Server*) usado no BOINC [Anderson 2004], um *middleware* para computação voluntária, que consegue distribuir cerca de $8,8$ milhões de tarefas por dia ($101,85$ tarefas por segundo) usando apenas um único computador de baixo custo. Com o uso de dois computadores adicionais, a sua capacidade aumenta para $23,6$ milhões de tarefas por dia ($273,14$ tarefas por segundo).

Fedak *at al.* [Fedak et al. 2010] construíram uma plataforma experimental para computação distribuída usando dispositivos de baixa capacidade conectados através de banda larga, chamada DSL-Lab, que oferece a possibilidade para pesquisadores realizarem experimentos em condições próximas àquelas que normalmente estão disponíveis com conexões domésticas com a Internet. Os resultados confirmam que é possível construir uma pilha completa de *software* em uma plataforma de design leve e de baixo custo sobre os dispositivos conectados em banda larga implementando gestão de recursos, eficiência energética, segurança e conectividade.

As estratégias propostas para o provisionamento OddCI para controlar o tamanho de instância e garantir que ele é adequado para a vazão requerida pelo cliente estão alinhadas com outras iniciativas de pesquisa. Aron e Chana propuseram um *framework* que oferece políticas de provisionamento para agendamento e alocação de recursos, e demonstraram que uma abordagem baseada no provisionamento de QoS é eficaz para minimizar o custo e o tempo de submissão de aplicações (*submission burst time*) [Aron e Chana 2012]. Rood e Lewis [Rood e Lewis 2009] estudaram a indisponibilidade freqüente e volátil de grades computacionais baseadas em recursos voluntários e usaram um modelo multi-estado para analisar um *log* de disponibilidade de máquinas baseado em dados coletados do Condor [Litzkow, Livny e Mutka 1988]. Partindo desse estudo, desenvolveram técnicas de predição para prever transições de recursos nos estados do modelo e, com base em tais previsões, propuseram técnicas de replicação de tarefas e escalonadores que são capazes de replicar as tarefas que

são mais prováveis de falhar, melhorando a eficiência da execução das aplicações.

Considerando contextos com recursos computacionais não dedicados, a previsão de disponibilidade dos dispositivos representa um aspecto relevante do provisionamento. A disponibilidade de recursos no *middleware* para grades computacionais Condor é modelada em 5 estados [Litzkow, Livny e Mutka 1988; Rood e Lewis 2009]: *disponível*, *usuário presente*, *limiar de CPU excedido*, *evicção de tarefa* ou *encerramento elegante* (*graceful shutdown*) e *indisponível*. Tais estados diferenciam os tipos de indisponibilidade refletindo as políticas que os donos dos recursos preferem (por exemplo, permitir o uso do recurso mesmo quando parte do processamento estiver sendo utilizada). Com base nesses estados e no histórico de disponibilidade dos recursos [Rood e Lewis 2009], usam preditores para análise de intervalos considerando os N dias anteriores no mesmo horário da previsão (*N-Day*) ou considerando as N horas anteriores ao horário da previsão (*N-Recent*). A forma de análise considera o número de transições do estado disponível para cada outro estado de indisponibilidade (*transactional*) e calculam a porcentagem de tempo que o recurso permanece em cada estado (*durational*), utilizando uma inferência sobre esses valores como a probabilidade do recurso mudar para o estado a seguir. Além disso, um esquema de ponderação que considera um peso igual, onde todas as transições possuem a mesma influência no comportamento futuro do recurso (*equal weighting*). Outro esquema tem ponderação de tempo, onde as transições que ocorreram mais próximas do horário previsto em N dias anteriores recebem um peso maior (*time weighting*) e, por fim, há a possibilidade de maior ponderação para a transição mais recente, não considerando o horário do dia (*frehness weighting*). Os resultados de maior acurácia de predição para o estado dos recursos entre os propostos foram de $77,3\%$ para a combinação *transitional/N-recent/freshness* (TRF) e $78,3\%$ para a combinação *transitional/N-Day/equal* (TDE). Essas duas combinações superaram outros preditores para recursos aplicáveis em grades computacionais como *Saturating and History Counter predictors* [Mickens e Noble 2006], *Multi-State and Single State Sliding Window predictors* [Dinda 2006] e *Ren Predictor* [Ren et al. 2007]. A abordagem TRF é semelhante à técnica de seleção por ranqueamento que usamos no Capítulo 5 mas requereu algumas simplificações para eliminar estados não naturais em alguns contextos nos quais os sistemas OddCI podem operar.

7.3 Uso de Recursos Não Convencionais em HTC

Considerando o uso de dispositivos não convencionais para a construção de infrasestruturas para executar aplicações HTC, podemos destacar quatro sistemas: o projeto BOINCOID [Boincoid 2011], o projeto Folding@home [Stanford 2011], o *Embbeded STB Cluster* [Neill et al. 2011], e o sistema TVGrid [Batista et al. 2007], o trabalho preliminar que levou à investigação abordada no Capítulo 6.

Neill *at al.* [Neill et al. 2011] investigam o uso de uma arquitetura de sistema heterogêneo que combina um *cluster* de computadores tradicionais com um conjunto integrado de *set-top-boxes* para executar aplicações paralelas. Os resultados experimentais também confirmam que a rede de banda larga de processadores embarcados é uma nova e promissora plataforma para uma variedade de aplicações paralelas com uso intensivo de processamento e armazenamento (*computationally intensive and data-intensive grid applications*) e já é capaz de proporcionar ganhos significativos de desempenho para algumas classes de aplicações Open MPI.

O projeto BOINCOID foi criado em 2008 e também endereça o uso de dispositivos não convencionais para execução de aplicações HTC com foco em sistemas baseados no sistema operacional Android. O seu objetivo principal é o porte da plataforma BOINC [Anderson 2004] para o Android, através da tradução do código original em C++ para a linguagem Java com a mantutenção do comportamento original. Esta iniciativa habilita a participação de um enorme contingente de dispositivos baseados no Android em projetos de computação voluntária como o Seti@Home [Anderson et al. 2002].

O Folding@home é um projeto de computação distribuída desenhado para realizar simulações moleculares para entender o dobramento de proteínas, má formações e doenças relacionadas. Iniciado em 2006, o projeto Folding@home começou a usar o tempo ocioso de consoles de videogames conectados à Internet para obter um desempenho na escala de PetaFLOPs [Folding@home 2011]. Essa experiência ratifica a tendência de usar dispositivos digitais emergentes e mostra a alta escalabilidade que tais dispositivos podem oferecer.

A proposta do TVGrid[1] tem por objetivo o aproveitamento, para computação em grade,

[1] A arquitetura proposta no TVGrid é baseada na patente de utilidade MU8600875-7 que foi inicialmente apresentada em "TVGrid: A Grid Architecture to use the idle resources on a Digital TV network" [Batista et al. 2007].

de recursos que seriam desperdiçados em uma rede de TV Digital, como banda de transmissão do canal e capacidade de processamento do receptor de TV Digital. Através de uma camada de *software* incorporada à rede de TV Digital e utilizando basicamente as tecnologias correntes do segmento - particularmente as tecnologias de *middleware* incorporadas pelos padrões ITU-T J.200, J.201 e J.202 - a abordagem TVGrid objetiva tornar possível utilizar a eventual infraestrutura ociosa para realizar processamento paralelo distribuído.

Partindo do princípio de que é possível modelar um sistema de televisão digital como um computador paralelo com quatro classes de elementos: as unidades de processamentos, a memória compartilhada, o sistema de entrada e saída e os barramentos que conectam esses elementos, o TVGrid apresenta uma arquitetura apta a executar aplicações de forma paralela nos receptores de TV Digital (Figura 7.1). São levados em consideração dois tipos de processadores: os mestres e os operários. Os processadores mestres só poderão escrever na memória compartilhada, enquanto que os processadores operários só poderão ler da memória compartilhada. Nesta arquitetura, o processador mestre é responsável por escrever na memória compartilhada as aplicações e os dados a serem processados pelos processadores operários. Os processadores operários acessam a memória compartilhada, lêem as aplicações e as executam. Qualquer dado necessário para a execução da aplicação será também lido da memória compartilhada. A saída do processamento é escrita no sistema de Entrada e Saída pelos processadores operários e lidas de lá pelo processador mestre.

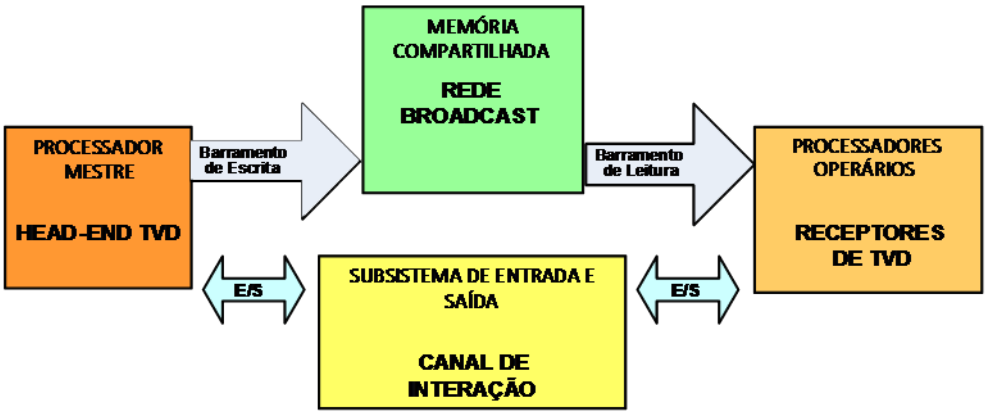


Figura 7.1: Os componentes de uma arquitetura de computação paralela representados como componentes de uma rede de TV Digital

As quatro classes de elementos descritas são representadas pelos seguintes componentes

na arquitetura do TVGrid:

- **Processador Mestre**: uma estação de TV equipada com um escalonador de tarefas é o componente responsável por distribuir as tarefas, através da rede de *broadcast*, para que os processadores operários as executem. Requer a integração de um Escalonador de Tarefas, disponibilizando os arquivos que compõem a tarefa: a aplicação Xlet e outros arquivos necessários, ao Gerador de Carrossel da estação de TV Digital para que sejam então serializados e injetados no multiplexador para que a tarefa possa ser transmitida junto com a programação do canal em questão.

- **Memória Compartilhada**: representada pelo meio físico de comunicação (terrestre, satélite ou cabo) utilizado pela estação de TV para transmissão em *broadcast* do sinal digital. Como o meio é compartilhado e a comunicação se dá de um para todos, apenas a estação de TV possui acesso de escrita nesse meio, mas os receptores recebem a programação do canal de TV (conteúdo audiovisual) multiplexada com dados (aplicações, informações de cotnrole, etc).

- **Sistema de Entrada e Saída**: O sistema de entrada e saída da arquitetura proposta é caracterizado pelo canal de interação bi-direcional (*full-duplex*) - comumente uma conexão com a Internet - que liga a estação de TV (processador mestre) e os receptores (processadores operários). No TVGrid, este canal de interação é utilizado basicamente para a transmissão do resultado processado pelos receptores para a estação de TV, para que o Escalonador de Tarefas faça o registro adequado de sua conclusão.

- **Processadores Operários**: Os processadores operários são os receptores de TV Digital capazes de executar as aplicações interativas multiplexadas junto à programação do canal - neste caso, aplicações Xlet compatíveis com GEM [ETSI 2004]. Esses receptores devem estar conectados ao canal de interação (sistema de Entrada e Saída), para que possam enviar à Estação de TV, ao término do processamento, o resultado de uma tarefa.

Limitações impostas por características particulares dos canais de comunicação que conectam os componentes de um sistema de TV Digital e a pela incapacidade, na forma nativa,

dos receptores de se comunicarem uns com os outros, torna a arquitetura do TVGrid mais adequada para executar aplicações BoT.

Na proposta do TVGrid [Batista et al. 2007] são discutidas duas lógicas possíveis para a implementação do escalonador de tarefas instalado na Estação de TV Digital, necessário para controlar o uso da infraestrutura do TVGrid. Tais abordagens, uma voltada para a execução de aplicações paramétricas e outra, um pouco mais complexa, para ser utilizada com aplicações BoT, estão resumidas a seguir:

- **Escalonador de Aplicações BoT**: Este escalonador requer o suporte de uma aplicação chamada de *trigger* (gatilho). A aplicação *trigger* é escrita no carrossel de objetos e carregada por todos os receptores que sintonizarem o canal utilizado pelo escalonador durante a transmissão do carrossel. Depois de ser carregada, a aplicação *trigger* é responsável pela execução de tarefas de uma aplicação BoT, copiando tanto a tarefa em si quanto os seus dados do carrossel de objetos para a memória local do receptor de TV Digital e finalmente executa a aplicação, armazenando o resultado também na sua memória local. Quando o programa finaliza sua execução, a aplicação *trigger* envia o resultado do processamento para a estação de TV Digital, limpa a memória e inicia o processo de execução de uma nova tarefa. Cada tarefa é transmitida em um *slot* que pode ser representado por um diretório em um sistema de arquivos. Assim, a aplicação *trigger* pode escolher a tarefa simplesmente escolhendo um *slot* aleatoriamente e executando sua tarefa correspondente. A mesma tarefa pode ser executada em paralelo por mais de um receptor. Essa redundância é necessária para garantir que todas as tarefas sejam realizadas (possibilidade estatística), apesar das possíveis falhas nos receptores, seu desligamento ou até mesmo a mudança de canal. O escalonador de tarefas deve ser responsável por identificar o recebimento do processamento replicado de tarefas, ignorando-as, ao mesmo tempo em que vai retirando da lista de tarefas da aplicação (ou *bag-of-tasks*) e substituindo no carrossel aquelas que já foram completadas. A aplicação termina quando a lista de tarefas fica vazia.

- **Escalonador de Aplicações Paramétricas**: este escalonador de tarefas é muito simples. Ele basicamente incluirá a aplicação paralela no carrossel de objetos de forma que o receptor de TV Digital sintonizado no canal utilizado pelo escalonador de tarefas

identifique que há uma aplicação multiplexada e que a mesma deve ser executada - para isso o receptor utiliza informações constantes na AIT (*Application Information Table*). É importante ressaltar que, como o carrossel de objetos se vale de um mecanismo que envia os mesmos dados repetidamente, qualquer receptor de TV Digital que sintonizar em um canal contendo aplicações disponíveis as irá carregar e executar, independente de quando ocorra a sintonia. Sempre que o escalonador receber de volta um resultado através da rede de interação, o mesmo deverá checar se os valores utilizados como entrada para o processamento enviado são diferentes de todos os valores utilizados em tarefas já executadas. Se essa condição for atendida, o resultado é armazenado como parte da saída geral da aplicação, caso contrário esse resultado é descartado. Quando um número suficiente de tarefas é completado, o escalonador pode iniciar a execução de outra aplicação utilizando a mesma estratégia, atualizando a aplicação no carrossel de objetos.

Ao adotar o conceito de escalonamento multi-nível, o OddCI-DTV torna-se mais flexível que a abordagem TV Grid com relação à gama de aplicações suportadas. A separação do processo de provisionamento e controle de recursos, realizada pelo *Controller*, da distribuição de tarefas, realizada pelo *Backend*, permite que controles fim-a-fim específicos de cada aplicação, incluindo os relativos à segurança, possam ser implementados facilmente. Além disso, o OddCI-DTV é mais transparente e requer uma menor participação da estação de TV na operacionalização de instâncias OddCI, o que pode, eventualmente, refletir em uma maior facilidade para implantação.

Capítulo 8

Conclusões e Trabalhos Futuros

8.1 Conclusões

Neste trabalho foram analisadas as razões que levam os fornecedores atuais de IaaS a imporem limites muito estritos sobre o número de recursos que qualquer cliente pode adquirir simultaneamente. Nossa avaliação utilizou um modelo de simulação para um provedor de IaaS, que é alimentado com uma carga de trabalho síntetica, o que permitiu a simulação de uma ampla variedade de cenários. A utili

zação de um modelo mais próximo da realidade nos pareceu a escolha mais adequada para este estudo. Para minimizar a complexidade do modelo e da falta de dados de campo, foram utilizadas técnicas como projeto de experimentos, para identificar as variáveis independentes mais importantes, e a varredura de parâmetros, permitindo a instanciação de uma grande variedade de configurações distintas. Foram obtidos resultados consistentes em todos os cenários simulados.

A análise mostra que é obrigatória a atribuição de um limite para a quantidade de recursos que podem ser alocados simultaneamente por qualquer usuário, a fim de manter a disponibilidade do serviço suficientemente elevada e a um custo razoável para o prestador. O valor real para esse limite vai variar de provedor para provedor dependendo de sua própria avaliação de onde situa-se o seu equilíbrio, mas os nossos resultados indicam que ele tende a não ser muito maior do que os valores atualmente praticados e que se enquadram no intervalo de algumas dezenas. Observou-se também que os usuários com perfis *Eventual* e *BoT* pressionam a capacidade mínima necessária e aumentam a ociosidade do sistema, aumentando

os custos operacionais do provedor. Além disso, mantidos o mesmo perfil da população e o mesmo valor de limite, a dinâmica do sistema independe da quantidade de usuários e, aparentemente, não constitui um contexto onde a economia de escala possa trazer melhorias substanciais.

Nosso estudo evidencia que quando a demanda dos usuários regulares é permanente e previsível, seu crescimento é benéfico para a lucratividade do provedor, posto que não impõe um risco de super provisionamento da infraestrutura. Desta forma, o lucro do provedor é negativamente afetado somente pela parcela da demanda que vem dos usuários eventuais, a qual pode resultar no crescimento da inatividade da infraestrutura, se não for controlada. Tal aspecto é especialmente ampliado quando os usuários eventuais são ávidos consumidores de recursos e fazem requisições pontuais muito grandes.

Os resultados ajudam a entender a necessidade do uso de um limite e como o seu impacto na lucratividade do provedor está diretamente relacionado com o padrão de utilização da população de usuários, nos fazendo concluir que algumas categorias de usuários/aplicações que se beneficiariam de uma elasticidade mais ampla, tendem a continuar sendo mal servidas se um modelo alternativo de provisionamento de recursos para provedores públicos de IaaS não emergir.

Neste sentido, os passos seguintes deste trabalho foram dedicados à investigação de formas alternativas para minimizar os custos envolvidos com o aumento da capacidade dos provedores públicos de computação na nuvem para lidar apropriadamente com a demanda de usuários eventuais ávidos por recursos, tais como aqueles que precisam executar grandes aplicações científicas *BoT*. Os custos associados com a ociosidade da infraestrutura são um dos principais obstáculos para a oferta de elasticidade em condições mais flexíveis, mesmo que ainda limitada, mas que permitam que classes de aplicações de uso intenso possam se beneficiar das vantagens do modelo de computação na nuvem. A descoberta, federação e revenda de recursos terceirizados pode representar um caminho promissor, pois se baseia no aproveitamento, sob demanda, de capacidade ociosa existente em contextos onde os custos de instalação e disponibilidade são absorvidos por terceiros.

Inspirados na filosofia "*Just in Time*" (JiT) da Toyota, nós propusemos as *Just in Time Clouds* ou *JiT Clouds*, uma abordagem alternativa para a construção de provedores de IaaS baseada na utilização de recursos terceirizados, onde os provedores apenas incorrem em

custos quando os recursos usados para prover a sua infraestrutura são demandados pelos seus clientes, permitindo uma ampliação de algumas ordens de magnitude no limite que precisa ser imposto aos clientes. Dessa forma, as *JiT Clouds* podem se apresentar como uma infraestrutura adequada para a execução de aplicações BoT de larga escala.

As *JiT Clouds* podem ser montadas sobre recursos que estejam distribuídos por todo o espectro de recursos terceirizados de baixa escala. Uma das missões do *JiT Provider* é descobrir e explorar o potencial dos recursos disponíveis alinhando-os com as necessidades das aplicações de clientes. Dependendo de suas características, os recursos terceirizados podem fornecer diferentes níveis de qualidade de serviço, elasticidade e escalabilidade. O nível de qualidade de serviço oferecido por um *JiT DC* é totalmente dependente do nível de qualidade de serviço suportado pelos recursos usados para montá-lo, o qual está relacionado ao padrão de granularidade, volatilidade e dispersão dos mesmos.

Quando os recursos estão concentrados em centros de dados e sua capacidade está localizada mais próxima do topo da magnitude que limita a baixa escala de recursos terceirizados, os níveis de serviço oferecidos são consistentes com os praticados pelos provedores tradicionais de computação na nuvem. Dessa forma, *JiT Clouds* baseadas em recursos de baixa granularidade, baixa volatilidade e baixa dispersão podem ser usadas para hospedar aplicações tipicamente suportadas por computação na nuvem. No outro extremo do espectro da escala, quando os recursos terceirizados são de grão pequeno e distribuídos, eles precisam ser agrupados e coordenados pelo *JiT Provider* para a sua exploração.

Para demonstrar a sua viabilidade, nós analisamos o potencial das *JiT Clouds* no seu cenário mais desafiador: considerando o uso de recursos computacionais de alta granularidade, alta volatilidade e alta dispersão para a composição de *JiT DCs* de alta vazão. Neste sentido e usando o conceito de redes de *broadcast*, foi proposta uma nova arquitetura, chamada de *Infraestrutura Computacional Distribuída Sob Demanda* ou OddCI, para construção de *JiT DCs* dinâmicos baseados em tais recursos computacionais através do uso de mecanismos específicos para a sua descoberta, alocação e coordenação. Nossos resultados de simulação mostram que, mesmo em cenários de altíssima volatilidade de nós autônomos e distribuídos geograficamente e sem o uso de algoritmos compensatórios ótimos, foi possível obter disponibilidade coletiva de dispositivos isolados para entregar vazão computacional com perdas máximas de 10% sob regimes de até 40% de volatilidade de nós, causada por

falhas ou abandonos voluntários. Entretanto, tal faixa de volatilidade já engloba uma série de cenários práticos no contexto estudado de TV Digital, por exemplo, os horários nobres, marcados pela transmissão de eventos de grande audiência, como jogos de futebol e novelas, e também os horários sem audiência, nos quais os receptores eventualmente ligados ficam permanentemente conectados em um mesmo canal.

No caso particular da aplicabilidade de sistemas OddCI para a descoberta, alocação e operação de *JiT DCs* dinâmicos, ficou evidenciado que a concorrência pelo uso do canal de *broadcast*, notadamente em contextos que envolvam a coordenação de muitas DCIs simultaneamente, requer a inclusão de mecanismos específicos em nível de controle de admissão e também na otimização da utilização dos recursos de comunicação de forma a permitir conciliar a qualidade do serviço prestado pelo provedor com os custos operacionais envolvidos.

A percepção intuitiva sobre a importância da estratégia de instanciação no processo de operação de sistemas OddCI foi devidamente comprovada. Através da análise dos resultados dos experimentos, fica bem evidente que recai sobre o *Controller* um papel fundamental no uso adequado dos recursos terceirizados e também no nível de cumprimento das demandas dos usuários. Por outro lado, também foi possível constatar que, adequadamente identificados e tratados, os aspectos de imprevisibilidade e volatilidade envolvidos no uso de recursos computacionais de redes de *broadcast* em *JiT DCs* dinâmicos podem ser contornados com a aplicação de algoritmos compensatórios.

Nosso entendimento dos sistemas OddCI foi consideravelmente ampliado com a construção do simulador OddCISim. Os desafios para o uso de redes de *broadcast* para a montagem de DCIs sob demanda que foram apenas levemente esboçados durante a definição da arquitetura OddCI puderam ser detalhados, refinados e, até mesmo, melhor compreendidos. Este entendimento ainda precisa ser ampliado com a investigação de estratégias de escalonamento e instanciação que funcionem bem em diversos cenários de recursos terceirizados e a prospecção de mecanismos que impeçam que a sobrecarga no esforço de coordenação possa tornar os *Controllers* um gargalo na escalabilidade de sistemas OddCI, especialmente quando manipularem redes de *broadcast* com uma grande quantidade de dispositivos. Entretanto, é possível minimizar alguns desses problemas com a adição de mecanismos mais inteligentes no controle de admissão e no planejamento de ações compensatórias que permitam distribuir melhor as instâncias ao longo do tempo de forma a evitar a sobreposição

desnecessária de mensagens de controle. Além disso, considerando que a oferta de grandes conjuntos de dispositivos computacionais por curtos espaço de tempo representa melhor o diferencial e vocação dos sistemas OddCI, é possível que as próprias características da demanda já atenuem esse efeito.

O uso da capacidade ociosa de processamento de muitos recursos computacionais distribuídos, tais como os dos receptores de TV digital já havia sido demonstrada antes, na proposta do TVGrid. Mas a generalização feita com a arquitetura OddCI e a construção de uma prova de conceito com a implementação da sistema OddCI-DTV sobre uma rede de TV Digital, a montagem de um testbed real e uma avaliação do seu desempenho mostraram não apenas a viabilidade dessa abordagem como também o fato de que a mesma é promissora.

Algumas limitações foram também entendidas. A primeira delas é que as aplicações BoT candidatas a rodar no OddCI-DTV devem ter uma restrição em foco: uso de pouca memória. Uma forma de verificar isso poderia ser uma homologação prévia por parte do *Provider*. Outras envolvem aspectos de implementação do PNA, que atua como um sistema operacional de alto nível para o escalonamento das aplicações e comunicação com o *Controller* e *Backend*. Em NCL/Lua ajustes de baixo nível ainda precisam ser feitos para proporcionar um maior desacoplamento do PNA com a aplicação BoT.

Na avaliação de desempenho de receptores de TV Digital de baixo custo para processamento de aplicações, foi observada uma diferença relevante de capacidade computacional quando comparados com dispositivos convencionais, mesmo os de baixa granularidade. Entretanto, acreditamos que essa perda não se constitui em uma limitação técnica irreparável mas, tão somente, um aspecto mercadológico e circunstancial, passível de ser contornado com facilidade caso uma demanda para dispositivos mais potentes seja criada. Basta sairmos um pouco do escopo da norma e da TV Digital aberta para encontraramos indícios consistentes de movimentos na direção de dispositivos mais poderosos. É o caso das TVs conectadas e receptores de TVs por assinatura, cujas funcionalidades e estão sendo permanentemente evoluídas em uma batalha pela preferência dos consumidores com efeitos imediatos na configuração dos equipamentos para poder suportá-las.

Atualmente, várias tecnologias já podem ser usadas para tornar possível a comunicação simultânea e unidirecional entre dispositivos digitais no modelo de um-para-muitos, característica fundamental do conceito de rede de *broadcast* evocado aqui. Da mesma forma,

também é bastante ampla a diversidade de dispositivos que podem ser alcançados por uma ou mais das tecnologias de transmissão mencionadas, desde computadores a equipamentos com fins mais específicos, tais como consoles de jogos, telefones celulares e receptores de TV digital. Alguns desses dispositivos menos tradicionais já provaram o seu potencial de uso para processamento distribuído em projetos de computação voluntária [Stanford 2011] [PS3 2011] [Boincoid 2011]. Tirando partido das funcionalidades já disponibilizadas sobre os dispositivos que implementam tais tecnologias ou complementando e/ou adaptando estas funcionalidades, é possível projetar implementações de Sistemas OddCI para diversos contextos.

Embora o foco desta pesquisa tenha sido a investigação da viabilidade técnica da abordagem proposta, há algumas evidências que apontam para a sua viabilidade do ponto de vista econômico.

Pela ótica dos proprietários dos recursos, um dos aspectos importantes a serem considerados é que a recompensa[1] percebida pelo fornecimento dos recursos excedentes seja superior aos custos envolvidos na própria cessão e permitam também um alívio nos custos que ocorrem independentemente dela. Ou seja, devem cobrir os custos de utilização (UC) e permitir a amortização, em algum grau, dos custos de disponibilidade associados com a manutenção de recursos excedentes, que continuam sendo de sua responsabilidade.

Um contexto onde isso é mais provável é quando os custos de disponibilidade dos recursos tercerizados excedentes já estão totalmente amortizados, tornando-os ainda mais atrativos para o seu aproveitamento em *JiT Clouds*. Neste sentido, um recurso é considerado *amortizado* se os seus custos fixos são totalmente cobertos, ao longo do tempo, pelo propósito original para o qual foi adquirido, considerando tanto os períodos de funcionamento pleno, quanto os períodos de ociosidade. Em outras palavras, um recurso é dito amortizado no caso de seu TCO não variar (ou variar pouco) devido à sua taxa de utilização.

Um dos custos de utilização mais importantes, notadamente no caso de recursos não convencionais, é a energia elétrica adicional consumida. Entretanto, quando consideramos receptores de TV Digital, tal incremento pode ser mínimo. De acordo com um estudo do Natural Resources Defense Council (NRDC) [Bloomberg 2011], dois terços do total de energia

[1] A análise de viabilidade comercial e negociação de preços de serviços, entre provedor e cliente, e preços de recursos, entre provedor e fornecedor, está fora do escopo desta pesquisa.

gasta por receptores de TV Digital é consumida quando eles não estão em uso. O problema é que os receptores estão sempre funcionando mesmo quando os usuários pensam que os desligaram. Em muitos casos, ativar o modo "*standby*" apenas escurece o relógio mas não coloca o receptor em um estado de menor consumo (*light-sleep*). Nós confirmamos esta condição em medições de consumo preliminares, que apontaram um aumento de apenas $1,14\%$ no consumo dos receptores usados em nossos testes quando processando aplicações.

Do ponto de vista do provedor da *JiT Cloud*, a vazão computacional ofertada deve ser atrativa e equilibrar preço e qualidade de serviço com o custo de operação da federação. Como o serviço prestado pode ser, potencialmente, muito mais elástico que os serviços ofertados pelos provedores atuais de computação na nuvem, o preço praticado por um *JiT Provider* pode ser balizado, no mínimo, com o preço cobrado pelos provedores de IaaS por recursos de capacidade similar. Note que, mesmo no caso de recursos não convencionais, dispositivos mais modernos já apresentam este tipo de equivalência com algumas classes de máquinas virtuais comercializadas, como visto no Capítulo 6.

Como o ônus do custo de disponibilidade dos recursos permanece como uma responsabilidade dos seus proprietários e o custo de utilização somente ocorre quando os recursos são efetivamente utilizados, o custo de coordenação da federação é o insumo mais relevante para o *JiT Provider*. Considerando que o custo de coordenação é uma função do tamanho da infraestrutura a ser gerenciada e não da forma com a mesma foi montada, possivelmente o custo de coordenação de uma *JiT Cloud* se manterá nos mesmos patamares apresentados por serviços baseados em infraestruturas próprias com a mesma categoria e tamanho. Entretanto, a coordenação da federação pode ser impactada pelo nível de serviço suportado pelos recursos envolvidos. Em especial, cenários de alta volatilidade podem apresentar níveis de falha que causem reflexos tanto nos custos operacionais da federação, pelo aumento do nível de redundância praticado, quanto na reputação do *JiT Provider*, que pode ser afetada por quedas na vazão entregue e por outras violações em SLAs.

Para algumas classes de aplicação, as *JiT Clouds* podem se apresentar como uma alternativa de maior valor agregado. É o caso em que a capacidade de prover grandes DCIs em regime de elasticidade extrema se torna um diferencial competitivo. Neste sentido, a escolha adequada pelo *JiT Provider* dos recursos terceirizados a serem federados em cada situação é fundamental. Por exemplo, no caso de recursos de uma rede de TV Digital, além da ca-

pacidade computacional requerida para os recursos, a observância de outros aspectos como audiência e horário de alocação, podem permitir o controle sobre a escala a ser atingida e a volatilidade a ser evitada.

De forma acessória, o uso de horários com maior ou menor audiência ou sem programação regular, popularmente chamados de "horário de chuvisco", também podem permitir acordos diferenciados pelo uso dos recursos em pauta. Quando observamos alguns indicadores mundiais de audiência televisiva [Wikipedia 2011], há diversos casos de eventos que conseguiram reunir centenas de milhões de espectadores simultaneamente e, na maioria dos países, há tipos específicos de programação local que concentram até 90% dos televisores ligados na sua faixa de horário. Tanto nos casos de eventos de grande audiência quanto nas situações em que o receptor em "*standby*" fica sintonizado em um canal, temos cenários de menor volatilidade, o que pode reduzir substancialmente o custo de coordenação[2]. A principal diferença entre os dois casos é a escala atingida, posto que os receptores deixados em "*standby*" não estão todos, necessariamente, sintonizados no mesmo canal como é o caso de eventos de grande audiência. Associadamente, as falhas em processamento causadas pelo encerramento da aplicação com a mudança do canal sintonizado, como previsto na maioria dos padrões de TDVI aberta, podem ser tratadas em receptores especialmente customizados para funcionamento em sistemas OddCI e também em TVs conectadas e receptores de TV por assinatura, normalmente baseados em sistemas proprietários.

Os principais resultados e contribuições deste trabalho, considerando as três questões de pesquisa que foram abordadas nesta pesquisa, são os seguintes:

Por que os provedores de nuvens públicas impõem limites que restringem a utilidade de seus serviços para clientes com aplicações BoT?

- Investigação das causas que levam os provedores públicos de computação na nuvem a impor um limite estrito na quantidade de recursos que um único usuário pode adquirir concomitantemente e análise de qual o impacto que eventuais aumentos no limite imposto apresentam sobre a lucratividade do provedor. Este resultado foi publicado no periódico *Elsevier Future Generation Computer Systems*: "Analyzing the Impact of Elasticity on the Profit of Cloud Computing Providers" [Costa et al. 2012e];

[2]Como visto no Capítulo 5, quando a volatilidade se encontra abaixo de 20%, a redundância máxima necessária para manter a vazão no nível requisitado é da ordem de 30%.

Como podemos servir adequadamente os usuários BoT em um cenário IaaS?

- Uma proposta de abordagem alternativa para montagem da infraestrutura computacional de um fornecedor de computação na nuvem com recursos de terceiros. A proposta introduz o conceito de *Just in Time Clouds*, cujos provedores apenas alocam os recursos quando eles são exigidos e somente durante o período que eles são necessários para os seus clientes. Isso elimina a necessidade de antecipar o planejamento de capacidade e exclui os custos associados ao excesso de provisionamento de recursos. Este resultado foi apresentado como poster na *3rd IEEE International Conference on Cloud Computing Technology and Science* (CloudCom 2011): "Just in Time Clouds: Enabling Highly-Elastic Public Clouds over Low Scale Amortized Resources" [Costa et al. 2011f]. Esta mesma abordagem foi submetida em 2010 na forma de um projeto para um edital da RNP/CTIC na área de Computação na Nuvem e foi aceito. Atualmente, este projeto nomeia o consórcio *JiT Clouds*, uma das duas redes de pesquisa atuais do CTIC na área de computação na nuvem, a qual é coordenada pela UFCG e congrega 17 instituições nacionais e internacionais em oito subgrupos de pesquisa;

É possivel construir JiT DCs nos cenários mais desafiadores, que envolvem recursos terceirizados de alta granularidade, alta volatilidade e alta dispersão?

- Uma proposta de uma nova arquitetura para computação distribuída que é ao mesmo tempo flexível e altamente escalável. Chamada de *OddCI - On-Demand Distributed Computing Infrastructure*, ela é suportada pela existência de um grande contingente de dispositivos que podem ser acessados simultaneamente através de uma rede de transmissão em *broadcast*. Este resultado foi publicado no *2nd Workshop on Many-Task Computing on Grids and Supercomputers* (MTAGS '09), realizado em conjunto com o Supercomputing 2009: "OddCI: On-demand Distributed Computing Infrastructure" [Costa et al. 2009];

- Implementação de um protótipo de sistema OddCI em um ambiente real de TV Digital para validação do conceito e obtenção de medições de campo. Um artigo descrevendo como o "testbed" foi construído e os resultados obtidos foi publicado na *IEEE/ACM International Conference on Grid Computing - GRID'12*: "OddCI-Ginga: A Platform for High Throughput Computing Using Digital TV Receivers" [Costa et al. 2012c];

- Um artigo consolidando esses e os outros resultados relacionados com a arquitetura OddCI foi publicado no periódico *Springer Journal of Grid Computing* em 2012: "Using Broadcast Networks to Create On-demand Extremely Large Scale High-throughput Computing Infrastructures" [Costa et al. 2012d].

8.2 Trabalhos Futuros

Há um desafio especial para a composição de modelos de negócio para os cenários de alta granularidade, conforme discutido na Seção 4.3.2, no qual o custo transacional e o baixo retorno monetário podem impor limites na parte inferior da escala dos recursos terceirizados que podem ser utilizados.

No entanto, em cenários específicos, o grão pode ser tão pequeno quanto possível. Este é o caso quando há um serviço aglutinador ("*glue service*") que absorve ou amortiza o custo transacional. No caso de dispositivos não convencionais como receptores de TV Digital e telefones celulares, eles podem ser agrupados e coordenados na escala apropriada pela estação de televisão e operadores de sistema de telefonia, respectivamente. Medidas de incentivo já existentes nesses contextos, bem como os canais correntes de faturamento e cobrança que podem ser totalmente reutilizados, reduzem ou eliminam os custos transacionais adicionais para o *JiT Provider*. Por exemplo, no caso de *JiT DCs* dinâmicos baseados em receptores de TV Digital, o proprietário do receptor pode ser recompensado na forma de créditos *pay-per-view*, representando uma recompensa de maior valor agregado do que o pagamento de quantidades muito pequenas de dinheiro. Através da compra de grandes lotes de créditos *pay-per-view*, o *JiT Provider* incrementa as vendas do operador de TV, ajudando no resultado operacional da emissora ou na cobertura dos custos da estrutura da sua rede de transmissão.

Uma frente de investigação futura poderia focar em modelos de negócio para *JiT Clouds* baseados no uso de agentes aglutinadores de recursos terceirizados de alta granularidade (como emissoras de TV, operadores de telefonia e provedores de banda larga e conteúdo etc.), que permitam conciliar:

- preços competitivos para os clientes de aplicações HTC em geral e BoT em particular;
- baixos custos operacionais para os *JiT Providers*;

- receita adicional e agregação de valor ao serviço original do agente aglutinador;
- mecanismos de incentivo que promovam a adesão dos proprietários dos recursos computacionais.

Outro trabalho futuro pode ser a implementação de novos mecanismos de predição e novas estratégias de escalonamento e provisionamento visando aumentar a efiência de coordenação do *Controller*. Para esta frente de investigação, podemos indicar dois aspectos iniciais a serem investigados:

- Prospecção de Mecanismos Escaláveis de Predição e Coordenação para o *Controller*: Desde que milhões de PNAs ativos podem estar, simultaneamente, enviando *heartbeat messages* para o *Controller*, mecanismos de hierarquização, otimização e distribuição de frequência de envio precisam ser incorporadas ao manuseio de tais mensagens para que as mesmas não representem um gargalo no sistema. Neste sentido, podem ser prospectados mecanismos eficientes e escaláveis de predição e coordenação que possam ser incorporados aos sistemas OddCI;
- Impactos das Estratégias de Provisionamento e Instanciação nos Custos do *Provider*: Em um primeiro momento, a seleção das estratégias pelo *Provider* e pelo *Controller* foi simplificada e direcionada para os aspectos de disponibilidade que o uso de dispositivos com maior ou menor taxa de volatilidade podiam trazer para a operacionalização das instâncias. Nesta nova frente de investigação podem ser tratados também os aspectos financeiros envolvidos na adoção de cada estratégia de escalonamento e provisionamento.

Em ambos os casos, as estratégias adicionais podem ter como característica comum um comportamento mais dinâmico, que envolva adaptabilidade às condições correntes de disponibilidade e custos da instância para decidir sobre a estratégia mais adequada a ser usada em cada *wakeup process*.

A abordagem OddCI exige canais de comunicação tanto em *broadcast* quanto bidirecionais para estar disponível. No entanto, o padrão de comunicação entre o aplicativo cliente pode seguir qualquer modelo (por exemplo, cliente/servidor, *peer-to-peer*), dependendo apenas das configurações de *firewall* do recurso computacional. Em princípio, as aplicações

mais adequadas para serem executados em sistemas OddCI não devem ser fortemente acopladas, tais como as que seguem os modelos MPI ou mesmo MapReduce. Aplicações com características de baixo acoplamento, tais como as que funcionam em plataformas de computação voluntária, como o BOINC, podem representar uma classe de aplicações que podem se beneficiar mais facilmente de sistemas OddCI. Um trabalho futuro interessante seria investigar como os sistemas OddCI podem interoperar com sistemas de computacão voluntária já estabelecidos.

Outros possíveis trabalhos futuros podem tratar outras questões que emergem no entorno do conceito das *JiT Clouds*:

- Como aferir e controlar os diferentes níveis de serviço suportados por cada fornecedor de recursos terceirizados a ser federado em uma *JiT Cloud*?
- Quais as classes de aplicação que são mais adequadas para *JiT Clouds*?
- Qual a relação entre o esforço despendido para a federação de infraestruturas baseadas em recursos terceirizados e a economia de custos obtida pelo provedor?

Bibliografia

[AB 2006]AB. *Agência Brasil: TV digital deve aumentar em 80 milhões número de aparelhos no país*. 2006. Disponível em: <http://www.agenciabrasil.gov.br/noticias/2006/07/06/materia.2006-07-06.4998754189/view>.

[ABNT 2009a]ABNT. *Televisao digital terrestre - Codificacao de dados e especificacoes de transmissao para radiodifusao digital - Parte 1*. 2009a. NBR 15606-1.

[ABNT 2009b]ABNT. *Televisao digital terrestre - Codificacao de dados e especificacoes de transmissao para radiodifusao digital - Parte 2*. 2009b. NBR 15606-2.

[ABNT 2009c]ABNT. *Televisao digital terrestre - Codificacao de dados e especificacoes de transmissao para radiodifusao digital - Parte 4*. 2009c. NBR 15606-4.

[Al-Fares, Loukissas e Vahdat 2008]AL-FARES, M.; LOUKISSAS, A.; VAHDAT, A. A scalable, commodity data center network architecture. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 38, p. 63–74, August 2008. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1402946.1402967>.

[Alliance 2011]ALLIANCE, C. S. *Cloud Security Alliance - CSA*. 2011. Disponível em: <http://cloudsecurityalliance.org/>.

[Altschul et al. 1990]ALTSCHUL, S. F. et al. Basic local alignment search tool. *J Molecular Biology*, v. 215, n. 3, p. 403–410, 1990.

[Amazon 2010]AMAZON. *Amazon Web Services (AWS)*. 2010. Disponível em: <http://aws.amazon.com>.

[Amazon 2011]AMAZON. *Amazon EC2 Spot Instances*. 2011. Disponível em: <http://aws.amazon.com/ec2/spot-instances>.

[Anandasivam, Buschek e Buyya 2009]ANANDASIVAM, A.; BUSCHEK, S.; BUYYA, R. A Heuristic Approach for Capacity Control in Clouds. In: *IEEE CEC 2009*. IEEE, 2009. p. 90–97. ISBN 978-0-7695-3755-9. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5210812>.

[Anderson 2004]ANDERSON, D. P. Boinc: A system for public-resource computing and storage. *Grid Computing, IEEE/ACM International Workshop on*, IEEE Computer Society, Los Alamitos, CA, USA, v. 0, p. 4–10, 2004. ISSN 1550-5510.

[Anderson et al. 2002]ANDERSON, D. P. et al. Seti@home: an experiment in public-resource computing. *Commun. ACM*, ACM, New York, NY, USA, v. 45, p. 56–61, November 2002. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/581571.581573>.

[Andrade et al. 2007]ANDRADE, N. et al. Automatic grid assembly by promoting collaboration in peer-to-peer grids. *J. Parallel Distrib. Comput.*, Academic Press, Inc., Orlando, FL, USA, v. 67, p. 957–966, August 2007. ISSN 0743-7315. Disponível em: <http://dl.acm.org/citation.cfm?id=1276523.1276643>.

[Andrzejak, Kondo e Anderson 2008]ANDRZEJAK, A.; KONDO, D.; ANDERSON, D. P. Ensuring collective availability in volatile resource pools via forecasting. In: *Proceedings of the 19th IFIP/IEEE international workshop on Distributed Systems: Operations and Management: Managing Large-Scale Service Deployment*. Berlin, Heidelberg: Springer-Verlag, 2008. (DSOM '08), p. 149–161. ISBN 978-3-540-85999-4. Disponível em: <http://dx.doi.org/10.1007/978-3-540-87353-2_12>.

[ARIB 2004]ARIB. *Association of Radio Industries and Businesses (ARIB): STD/B23 V1.1 Application Execution Engine Platform for Digital Broadcasting (English Translation)*. 2004. Disponível em: <http://www.arib.or.jp/english/html/overview/doc/6-STD-B23v1_1-E1.pdf>.

[Armbrust et al. 2009]ARMBRUST, M. et al. *Above the Clouds : A Berkeley View of Cloud Computing*. 2009. 1–25 p.

[Arnold e Gosling 1996]ARNOLD, K.; GOSLING, J. *The Java Programming Language*. Addison Wesley, 1996. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1240605>.

[Aron e Chana 2012]ARON, R.; CHANA, I. Formal QoS Policy Based Grid Resource Provisioning Framework. *Journal of Grid Computing*, Springer Netherlands, v. 10, p. 249–264, 2012. ISSN 1570-7873. 10.1007/s10723-012-9202-y. Disponível em: <http://dx.doi.org/10.1007/s10723-012-9202-y>.

[Badger et al. 2011]BADGER, L. et al. *Cloud Computing Synopsis and Recommendations*. [S.l.], maio 2011.

[Barroso e Hölzle 2007]BARROSO, L. A.; HöLZLE, U. The Case for Energy-Proportional Computing. *Computer*, v. 40, n. 12, p. 33–37, dez. 2007. ISSN 0018-9162. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4404806>.

[Batista C. E. C. F. 2006]BATISTA C. E. C. F., . Tv digital - java na sala de estar. *Mundo Java*, Mundo Java, n. 17, 2006.

[Batista et al. 2007]BATISTA, C. E. C. F. et al. Tvgrid: A grid architecture to use the idle resources on a digital tv network. In: *Proc. 7th IEEE International Symposium on Cluster Computing and the Grid (The Latin America Grid Workshop)*. Rio de Janeiro, Brazil: [s.n.], 2007. p. 823–828.

[Bell e LaPadula 1976]BELL, D. E.; LAPADULA, L. J. *Secure Computer Sytems: United Exposition and Multics Interpretation*. [S.l.], 1976.

[Bitcurrent 2011]BITCURRENT. *Bitcurrent Team*. 2011. Disponível em: <http://www.bitcurrent.com/>.

[Bloomberg 2011]Bloomberg. *Stop Cable Boxes From Draining NationÕs Power Supply: View*. 2011. Disponível em: <http://www.bloomberg.com/news/2011-07-11/stop-cable-boxes-from-draining-the-nation-s-power-supply-view.html>.

[BOB 2008]BOB. *Beijing Olympics Blog: Record 4.7 billion Television Viewers Watched Beijing Olympic Games 2008*. 2008. Disponível em: <http://beijing-olympics-blog.blogspot.com/2008/10/record-47-billion-television-viewers.html>.

[Boesgaard e Zenner 2007]BOESGAARD, M.; ZENNER, E. Protecting online transactions with unique embedded key generators. In: *Proceedings of the The Second International Conference on Availability, Reliability and Security*. Washington, DC, USA: IEEE Computer Society, 2007. p. 663–669. ISBN 0-7695-2775-2. Disponível em: <http://dl.acm.org/citation.cfm?id=1249254.1250580>.

[Boincoid 2011]BOINCOID. *Boincoid - An Android Port of the Boinc Platform*. 2011. Disponível em: <http://boincoid.sourceforge.net>.

[Buyya, Ranjan e Calheiros 2010]BUYYA, R.; RANJAN, R.; CALHEIROS, R. N. Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. *Network*, Springer, v. 6081/2010, n. LNCS 6081, p. 20, 2010. Disponível em: <http://arxiv.org/abs/1003.3920>.

[Chandra e Weissman 2009]CHANDRA, A.; WEISSMAN, J. Nebulas: using distributed voluntary resources to build clouds. In: *Proceedings of the 2009 conference on Hot topics in cloud computing*. Berkeley, CA, USA: USENIX Association, 2009. (HotCloud'09). Disponível em: <http://dl.acm.org/citation.cfm?id=1855533.1855535>.

[Cirne et al. 2006]CIRNE, W. et al. Labs of the World, Unite!!! *Journal of Grid Computing*, v. 4, n. 3, p. 225–246, 2006. Disponível em: <http://dx.doi.org/10.1007/s10723-006-9040-x>.

[Cirne et al. 2003]CIRNE, W. et al. *Running Bag-of-Tasks applications on computational grids: the MyGrid approach*. IEEE, 2003. 407–416 p. ISBN 0-7695-2017-0. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1240605>.

[CloudScaling 2009]CLOUDSCALING. *Amazon's EC2 Generating 220M+ Annually*. 2009. Disponível em: <http://cloudscaling.com/ blog/cloud-computing/amazons-ec2-generating-220m-annually>.

[CloudStandards 2011]CloudStandards. *Cloud Standards - CS*. 2011. Disponível em: <http://cloud-standards.org>.

[Coffman Jr. e Wood 1966]COFFMAN JR., E. G.; WOOD, R. C. Interarrival statistics for time sharing systems. *Commun. ACM*, ACM, New York, NY, USA, v. 9, n. 7, p. 500–503, jul. 1966. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/365719.365961>.

[Costa et al. 2012c]COSTA, R. et al. Oddci-ginga: A platform for high throughput computing using digital tv receivers. In: *IEEE/ACM International Conference on Grid Computing - GRID'12*. Los Alamitos, CA, USA: IEEE Computer Society, 2012c. (GRID'12, v. 0), p. 155–163. ISSN 1550-5510.

[Costa et al. 2011e]COSTA, R. et al. Uma análise do impacto da elasticidade no lucro de provedores de computação na nuvem (in press). *Revista Brasileira de Redes e Sistemas Distribuídos (RB-RESD)*, Sociedade Brasileira de Computação, v. 4, n. 1, 2011e.

[Costa et al. 2012d]COSTA, R. et al. Using broadcast networks to create on-demand extremely large scale high-throughput computing infrastructures. *Journal of Grid Computing*, Springer Netherlands, v. 10, p. 419–445, 2012d. ISSN 1570-7873. Disponível em: <http://dx.doi.org/10.1007/s10723-012-9229-0>.

[Costa et al. 2009]COSTA, R. et al. Oddci: on-demand distributed computing infrastructure. In: *2nd Workshop on Many-Task Computing on Grids and Supercomputers*. Portland, Oregon: ACM, 2009. v. 16, p. 1–10.

[Costa et al. 2012e]COSTA, R. et al. Analyzing the impact of elasticity on the profit of cloud computing providers. *Future Generation Computer Systems (In Press)*, Elsevier Netherlands, 2012e.

[Costa et al. 2011f]COSTA, R. et al. Just in Time Clouds: Enabling Highly-Elastic Public Clouds over Low Scale Amortized Resources. In: *3rd IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2011)*. Athens - Greece: [s.n.], 2011f.

[Costa et al. 2013]COSTA, R. et al. Sobre o Uso de Dispositivos de Alta Granularidade, Alta Volatilidade e Alta Dispersão em Just in Time Clouds. In: *XXXI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2012)*. Brasília - DF: [s.n.], 2013.

[D'Anna et al. 2003]D'ANNA, L. et al. *Self-protecting mobile agents obfuscation report*. [S.l.], 2003.

[Dean e Ghemawat 2008]DEAN, J.; GHEMAWAT, S. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, ACM, New York, NY, USA, v. 51, p. 107–113, January 2008. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/1327452.1327492>.

[Deavours et al. 2002]DEAVOURS, D. et al. The Mobius framework and its implementation. *IEEE Transactions on Software Engineering*, v. 28, n. 10, p. 956–969, out. 2002. ISSN 0098-5589. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1041052>.

[Dinda 2006]DINDA, P. A. Design, implementation, and performance of an extensible toolkit for resource prediction in distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, IEEE Computer Society, Los Alamitos, CA, USA, v. 17, n. 2, p. 160–173, 2006. ISSN 1045-9219.

[DVB 2011]DVB. *Digital Video Broadcasting - The Global Standard for Digital Television*. 2011. Disponível em: <http://www.dvb.org>.

[Eduardo, Leite e Rodrigues 2005]EDUARDO, L.; LEITE, C.; RODRIGUES, R. F. Flextv uma proposta de arquitetura de middleware para o sistema brasileiro de tv digital. *Revista de Engenharia de Computação e Sistemas Digitais*, Citeseer, v. 2, p. 29–49, 2005.

[ETSI 2004]ETSI. *ETSI Standard. TS 102 819: Globally Executable MHP (GEM)*. 2004. Disponível em: <http://webapp.etsi.org/workprogram/Report_WorkItem.asp?WKI_ID=19737>.

[Eucalyptus 2011]EUCALYPTUS. *Eucalyptus Cloud Computing Software*. 2011. Disponível em: <http://http://www.eucalyptus.com/>.

[Evangelinos e Hill 2008]EVANGELINOS, C.; HILL, C. N. Cloud Computing for parallel Scientific HPC Applications: Feasibility of Running Coupled Atmosphere-Ocean Climate Models on Amazon's EC2. In: *Cloud Computing and Its Applications*. [s.n.], 2008. Disponível em: <http://www.cca08.org/speakers/evangelinos.php>.

[Fedak et al. 2010]FEDAK, G. et al. DSL-Lab: a platform to experiment on domestic broadband internet. In: *9th International Symposium on Parallel and Distributed Computing (ISPDC'2010)*. Istanbul, Turkey: [s.n.], 2010.

[Feitelson 2009]FEITELSON, D. G. *Workload Modeling for Computer Systems Performance Evaluation*. 0.30. ed. Hebrew University of Jerusalem (Online Book), 2009. Disponível em: <http://www.cs.huji.ac.il/f̃eit/wlmod/>.

[Filho, Leite e Batista 2007]FILHO, G. L. d. S.; LEITE, L. E. C.; BATISTA, C. E. C. F. Ginga-J: the procedural middleware for the Brazilian digital TV system. *Journal of the Brazilian Computer Society*, scielo, v. 12, p. 47 – 56, 03 2007. ISSN 0104-6500.

[Folding@home 2011]FOLDING@HOME. *Folding@home Petaflop Barrier Crossed*. 2011. Disponível em: <http//blog.us.playstation.com/2007/09/19/foldinghome-petaflop-barrier-crossed>.

[Force 2011]FORCE, D. M. T. *Distributed Management Task Force - DMTF*. 2011. Disponível em: <http://http://dmtf.org>.

[Force 2011]FORCE, D. M. T. *Open Virtualization Format (OVF)*. 2011. Disponível em: <http://http://dmtf.org/standards/ovf>.

[Foster et al. 2008]FOSTER, I. et al. Cloud computing and grid computing 360-degree compared. In: *Grid Computing Environments Workshop, 2008. GCE '08*. [S.l.: s.n.], 2008. p. 1 –10.

[Fox 2011]FOX, A. Computer science. cloud computing–what is in it for me as a scientist? *Science*, American Association for the Advancement of Science, v. 331, n. 6016, p. 406–407, 2011. Disponível em: <http://www.sciencemag.org/cgi/doi/10.1126/science.1198981>.

[Fox 2002]FOX, B. Digital TV Rollout. *IEEE Spectrum*, IEEE, v. 38, n. 2, p. 65–67, 02 2002.

[Francois, State e Festor 2007a]FRANCOIS, J.; STATE, R.; FESTOR, O. Botnets for scalable management. In: *Proceedings of the Distributed systems: operations and management 18th IFIP/IEEE international conference on Managing virtualization of networks and services*. Berlin, Heidelberg: Springer-Verlag, 2007a. (DSOM07), p. 1–12. ISBN 3-540-75693-0, 978-3-540-75693-4. Disponível em: <http://dl.acm.org/citation.cfm?id=1783374.1783376>.

[Francois, State e Festor 2007b]FRANCOIS, J.; STATE, R.; FESTOR, O. Malware models for network and service management. In: *Proceedings of the 1st international conference on Autonomous Infrastructure, Management and Security: Inter-Domain Management*. Berlin, Heidelberg: Springer-Verlag, 2007b. (AIMS 07), p. 192–195. ISBN 978-3-540-72985-3. Disponível em: <http://dx.doi.org/10.1007/978-3-540-72986-0_23>.

[Francois, State e Festor 2008]FRANCOIS, J.; STATE, R.; FESTOR, O. Towards malware inspired management frameworks. In: *Network Operations and Management Symposium (NOMS)*. Salvador, Bahia: IEEE, 2008. p. 105–112.

[Freeman e Lessiter 2003]FREEMAN, J.; LESSITER, J. Using Attitude Based Segmentation to Better Understand Viewers' Usability Issues with Digital and Interactive TV. In: MASTHOF, J.; GRIFFITHS, R.; PEMBERTON, L. (Ed.). *Proceedings of the 1st European Conference on Interactive Television: from Viewers to Actors?* [s.n.], 2003. p. 19–27. Disponível em: <http://www.brighton.ac.uk/interactive/euroitv/Papers/Paper3.pdf>.

[Golden 2009]GOLDEN, B. *The Case Against Cloud Computing*. 2009. Disponível em: <http://www.cio.com/article/477473/The_Case_Against_Cloud_Computing_Part_One>.

[Greenberg et al. 2008]GREENBERG, A. et al. The cost of a cloud: Research Problem in Data Center Networks. *ACM SIGCOMM Computer Communication Review*, v. 39, n. 1, p. 68, dez. 2008. ISSN 01464833. Disponível em: <http://portal.acm.org/citation.cfm?doid=1496091.1496103>.

[GreenGrid 2010]GREENGRID. *The Green Grid*. 2010. Disponível em: <http://www.thegreengrid.org>.

[Hey e Trefethen 2003]HEY, A. J. G.; TREFETHEN, A. E. The Data Deluge: An e-Science Perspective. In: ____. *Grid Computing Making the Global Infrastructure a Reality*. Wiley and Sons, 2003. (2003, January), cap. 36, p. 809–824. Disponível em: <http://eprints.ecs.soton.ac.uk/7648/>.

[Hogan et al. 2011]HOGAN, M. et al. *NIST Cloud Computing Standards Roadmap*. [S.l.], julho 2011.

[Iosup et al. 2008]IOSUP, R. et al. *An early performance analysis of cloud computing services for scientific computing*. [S.l.], 2008. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.174.7949>.

[ISO/IEC 1994]ISO/IEC. *ISO/IEC 13818.2. MPEG Committee International Standard: Generic Coding of Moving Pictures and Associated Audio Information: Video. ISOMEG*. 1994. Disponível em: <http://www.iso.org/iso/catalogue_detail.htm?csnumber=31537>.

[ISO/IEC 1998]ISO/IEC. *ISO/IEC TR 13818.6. Information technology: Generic coding of moving pictures and associated audio information. Part 6: Extensions for DSM/CC*. 1998. Disponível em: <http://www.iso.org/iso/catalogue_detail.htm?csnumber=25039>.

[ITVW 2011]ITVW. *The Interactive TV Web: The Java TV Tutorial*. 2011. Disponível em: <http://www.interactivetvweb.org/tutorials/javatv>.

[Jain 1991]JAIN, R. *The Art of Computer Systems Performance Analysis*. John Wiley and Sons, 1991. 716 p. ISBN 0471503363. Disponível em: <http://books.google.com/books?id=eOR0kJjgMqkC&pgis=1>.

[Jung, Krishnamurthy e Rabinovich 2002]JUNG, J.; KRISHNAMURTHY, B.; RABINOVICH, M. *Flash crowds and denial of service attacks*. New York, New York, USA: ACM Press, 2002. 293 p. ISBN 1581134495. Disponível em: <http://portal.acm.org/citation.cfm?doid=511446.511485>.

[Juve et al. 2009]JUVE, G. et al. Scientific workflow applications on amazon ec2. *2009 5th IEEE International Conference on EScience Workshops*, Ieee, p. 59–66, 2009. Disponível em: <http://arxiv.org/abs/1005.2718>.

[Keahey 2010]KEAHEY, K. *Another Barrier Goes Down*. 2010. Disponível em: <http://scienceclouds.org/blog/>.

[Keahey, Doering e Foster 2004]KEAHEY, K.; DOERING, K.; FOSTER, I. From sandbox to playground: Dynamic virtual environments in the grid. In: *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*. Washington, DC, USA: IEEE Computer Society, 2004. (GRID '04), p. 34–42. ISBN 0-7695-2256-4. Disponível em: <http://dx.doi.org/10.1109/GRID.2004.32>.

[Kepler 2010]KEPLER. *Kepler Project: MD5 Cryptographic Library for Lua*. 2010. Disponível em: <//www.keplerproject.org/md5/>.

[Kirby et al. 2010]KIRBY, G. et al. An approach to ad hoc cloud computing. *Arxiv preprint arXiv*, 2010. Disponível em: <http://arxiv.org/abs/1002.4738>.

[Lagar-Cavilla et al. 2009]LAGAR-CAVILLA, H. A. et al. Snowflock: rapid virtual machine cloning for cloud computing. In: *Proceedings of the 4th ACM European conference on Computer systems*. New York, NY, USA: ACM, 2009. (EuroSys '09), p. 1–12. ISBN 978-1-60558-482-9. Disponível em: <http://doi.acm.org/10.1145/1519065.1519067>.

[Landry, Malouin e Oral 1983]LANDRY, M.; MALOUIN, J.-L.; ORAL, M. Model validation in operations research. *European Journal of Operational Research*, v. 14, n. 3, p. 207 – 220, 1983. ISSN 0377-2217. ¡ce:title¿Methodology, Risk and Personnel¡/ce:title¿. Disponível em: <http://www.sciencedirect.com/science/article/pii/0377221783902576>.

[Lee 2010]LEE, C. A. A perspective on scientific cloud computing. In: *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. New York, NY, USA: ACM, 2010. (HPDC '10), p. 451–459. ISBN 978-1-60558-942-8. Disponível em: <http://doi.acm.org/10.1145/1851476.1851542>.

[Li et al. 2009]LI, X. et al. The Method and Tool of Cost Analysis for Cloud Computing. *2009 IEEE International Conference on*

Cloud Computing, Ieee, p. 93–100, set. 2009. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5284157>.

[Litzkow, Livny e Mutka 1988]LITZKOW, M.; LIVNY, M.; MUTKA, M. Condor - a hunter of idle workstations. In: *Proceedings of the 8th International Conference of Distributed Computing Systems*. IEEE Comput. Soc. Press, 1988. p. 104–111. ISBN 0-8186-0865-X. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=12507>.

[Lunt, Neumann e al. 1998]LUNT, T. F.; NEUMANN, P. G.; AL., D. D. et. *Security policy and policy interpretation for a class A1 multilevel secure*. Menlo Park, CA, 1998.

[May 1999]MAY, M. *Idle Computing Resources as Micro-Currencies - Bartering CPU Time for Online Content. Citeseer. WebNet*. 1999. Disponível em: <Citeseer. WebNet>.

[McLaughlin 2004]MCLAUGHLIN, L. Bot software spreads, causes new worries. *IEEE Distributed Systems Online*, IEEE Computer Society, Los Alamitos, CA, USA, v. 5, 2004. ISSN 1541-4922.

[Menascé e Ngo 2009]MENASCé, D. A.; NGO, P. Understanding Cloud Computing: Experimentation and Capacity Planning. In: *2009 Computer Measurement Group Conference*. [S.l.: s.n.], 2009. p. 11.

[Mickens e Noble 2006]MICKENS, J. W.; NOBLE, B. D. Improving distributed system performance using machine availability prediction. *SIGMETRICS Perform. Eval. Rev.*, ACM, New York, NY, USA, v. 34, n. 2, p. 16–18, set. 2006. ISSN 0163-5999. Disponível em: <http://doi.acm.org/10.1145/1168134.1168143>.

[Microsystems 2011]MICROSYSTEMS, S. *Java Technology in Digital TV*. 2011. Disponível em: <http://java.sun.com/products/javatv>.

[Mieritz e Kirwin 2005]MIERITZ, L.; KIRWIN, B. *Defining Gartner Total Cost of Ownership*. 2005. Disponível em: <http://www.gartner.com/DisplayDocument?id=487157&ref=g_sitelink>.

[Miser 1993]MISER, H. J. A foundational concept of science appropriate for validation in operational research. *European Journal of Operational Research*, v. 66, n. 2, p. 204 – 215, 1993. ISSN 0377-2217. ¡ce:title¿Model Validation¡/ce:title¿. Disponível em: <http://www.sciencedirect.com/science/article/pii/037722179390313C>.

[Morris e Chaigneau 2005]MORRIS, S.; CHAIGNEAU, A. S. *Interactive TV Standards: A Guide to MHP, OCAP, and JavaTV*. Focal Press, 2005. ISBN 0240806662. Disponível em: <http://portal.acm.org/citation.cfm?id=1207386>.

[NCBI 2011]NCBI. *National Center for Biotechnology Information (NCBI): The Basic Local Alignment Search Tool (BLAST)*. 2011. Disponível em: <http//blast.ncbi.nlm.nih.gov/Blast.cgi>.

[Neill et al. 2011]NEILL, R. et al. Embedded processor virtualization for broadband grid computing. In: *Proceedings of the 2011 IEEE/ACM 12th International Conference on Grid Computing*. Washington, DC, USA: IEEE Computer Society, 2011. (GRID '11), p. 145–156. ISBN 978-0-7695-4572-1. Disponível em: <http://dx.doi.org/10.1109/Grid.2011.27>.

[Neustar 2011]NEUSTAR. *Neustar Webmetrics*. 2011. Disponível em: <http//www.webmetrics.com/>.

[Oberheide, Cooke e Jahanian 2008]OBERHEIDE, J.; COOKE, E.; JAHANIAN, F. Exploiting live virtual machine migration. In: *Black Hat DC Briefings*. Washington DC: [s.n.], 2008.

[Oliveira, Baião e Mattoso 2011]OLIVEIRA, D. de; BAIãO, F.; MATTOSO, M. Migração de experimentos científicos para a nuvem. *Revista Horizontes*, Sociedade Brasileira de Computação, n. Abril 2011, 2011.

[Oliveira, Lopes e Silva 2002]OLIVEIRA, L.; LOPES, L.; SILVA, F. P3: Parallel peer to peer an internet parallel programming environment. *Web Engineering and PeertoPeer Computing*, p. 274–288, 2002. Disponível em: <http://dx.doi.org/10.1007/3-540-45745-3_25>.

[OpenNebula 2011]OpenNebula. *Open Nebula: The Open Source Toolkit for Cloud Computing*. 2011. Disponível em: <http://http://opennebula.org/>.

[OpenStack 2011]OpenStack. *Open Stack: Cloud Software*. 2011. Disponível em: <http://http://www.openstack.org/>.

[Patel e Shah 2005]PATEL, C. D.; SHAH, A. *Cost Model for Planning, Development and Operation of a Data Center*. [S.l.], june 2005.

[Peng 2002]PENG, C. Digital television applications. *Technology*, Citeseer, 2002.

[PS3 2011]PS3. *Folding@home PS3 FAQ*. 2011. Disponível em: <http//folding.stanford.edu/English/FAQ-PS3>.

[Raicu et al. 2008]RAICU, I. et al. Toward loosely coupled programming on petascale systems. In: *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. Piscataway, NJ, USA: IEEE Press, 2008. (SC '08), p. 22:1–22:12. ISBN 978-1-4244-2835-9. Disponível em: <http://dl.acm.org/citation.cfm?id=1413370.1413393>.

[Raicu et al. 2007]RAICU, I. et al. Falkon: a Fast and Light-weight tasK executiON framework. In: *SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing*. New York, NY, USA: ACM, 2007. p. 1–12. ISBN 978-1-59593-764-3. Disponível em: <http://dx.doi.org/10.1145/1362622.1362680>.

[Ren et al. 2007]REN, X. et al. Prediction of resource availability in fine-grained cycle sharing systems empirical evaluation. *Journal of Grid Computing*, Kluwer Academic Publishers, v. 5, p. 173–195, 2007. ISSN 1570-7873. Disponível em: <http://dx.doi.org/10.1007/s10723-007-9077-5>.

[Rightscale 2011]RIGHTSCALE. *Rightscale Cloud Management Platrform*. 2011. Disponível em: <http://www.rightscale.com>.

[Rimal, Choi e Lumb 2009]RIMAL, B.; 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*. [S.l.: s.n.], 2009. p. 44 –51.

[Rochwerger et al. 2009]ROCHWERGER, B. et al. The reservoir model and architecture for open federated cloud computing. *IBM J. Res. Dev.*, IBM Corp., Riverton, NJ, USA, v. 53, p. 535–545, July 2009. ISSN 0018-8646. Disponível em: <http://dl.acm.org/citation.cfm?id=1850659.1850663>.

[Rood e Lewis 2009]ROOD, B.; LEWIS, M. Grid Resource Availability Prediction-Based Scheduling and Task Replication. *Journal of Grid Computing*, Springer Netherlands, v. 7, p. 479–500, 2009. ISSN 1570-7873. 10.1007/s10723-009-9135-2. Disponível em: <http://dx.doi.org/10.1007/s10723-009-9135-2>.

[Sargent 1998]SARGENT, R. Verification and validation of simulation models. In: *Simulation Conference (WSC), Proceedings of the 1998 Winter Simulation Conference*. [S.l.: s.n.], 1998. p. 166–183. ISSN 0891-7736.

[Sarmenta 2001]SARMENTA, L. F. G. Sabotage-tolerance mechanisms for volunteer computing systems. In: *Proceedings of the 1st International Symposium on Cluster Computing and the Grid*. Washington, DC, USA: IEEE Computer Society, 2001. (CCGRID '01), p. 337–. ISBN 0-7695-1010-8. Disponível em: <http://dl.acm.org/citation.cfm?id=560889.792320>.

[Schellhorn et al. 2002]SCHELLHORN, G. et al. Verified formal security models for multiapplicative smart cards. *Computer Security*, IOS Press, Amsterdam, The Netherlands, v. 10, p. 339–367, December 2002. ISSN 0926-227X. Disponível em: <http://dl.acm.org/citation.cfm?id=773069.773072>.

[Scripps 2011]SCRIPPS. *FightAIDS@home - The Scripps Research Institute (SRI)*. 2011. Disponível em: <http://fightaidsathome.scripps.edu>.

[Sens 2010]SENS, P. Byzantine failure detection for dynamic distributed systems. *Distributed Computing*, 2010. Disponível em: <http://en.scientificcommons.org/55302834>.

[Sevior, Fifield e Katayama 2010]SEVIOR, M.; FIFIELD, T.; KATAYAMA, N. Belle monte-carlo production on the amazon ec2 cloud. *Journal of Physics: Conference Series*, v. 219, n. 1, p. 012003, abr. 2010. ISSN 1742-6596. Disponível em: <http://stacks.iop.org/1742-6596/219/i=1/a=012003>.

[Shiers 2010]SHIERS, J. D. Can clouds replace grids? will clouds replace grids? *Journal of Physics: Conference Series*, v. 219, n. 6, p. 062026, 2010. Disponível em: <http://stacks.iop.org/1742-6596/219/i=6/a=062026>.

[Simmons, McCloskey e Lutfiyya 2007]SIMMONS, B.; MCCLOSKEY, A.; LUTFIYYA, H. Dynamic provisioning of resources in data centers. In: *Proceedings of the Third International Conference on Autonomic and Autonomous Systems*. Washington, DC, USA: IEEE Computer Society, 2007. p. 40–. ISBN 0-7695-2859-5. Disponível em: <http://dl.acm.org/citation.cfm?id=1270386.1270808>.

[Stanford 2011]STANFORD. *Stanford University: Folding@home Distributed Computing*. 2011. Disponível em: <http//folding.stanford.edu>.

[Stanoevska-Slabeva e Wozniak 2010]STANOEVSKA-SLABEVA, K.; WOZNIAK, T. Cloud basics - an introduction to cloud computing. *Grid and Cloud Computing*, Springer Berlin Heidelberg, p. 47–61, 2010. Disponível em: <http://dx.doi.org/10.1007/978-3-642-05193-7_4>.

[Talby 2006]TALBY, D. *User Modeling of Parallel Workloads by User Modeling of Parallel Workloads*. Hebrew University of Jerusalem (PhD Thesis), 2006. Disponível em: <http://www.cs.huji.ac.il/labs/parallel/stud/Talby-PhD.pdf>.

[Thain, Tannenbaum e Livny 2006]THAIN, D.; TANNENBAUM, T.; LIVNY, M. How to measure a large open-source distributed system: Research articles. *Concurr. Comput. : Pract. Exper.*, John Wiley and Sons Ltd., Chichester, UK, v. 18, p. 1989–2019, December 2006. ISSN 1532-0626. Disponível em: <http://dl.acm.org/citation.cfm?id=1182902.1182908>.

[Toyota Motor Co 2011]Toyota Motor Co. *"Just in Time", Toyota Production System (TPS)*. 2011. Disponível em: <http://www2.toyota.co.jp/en/vision/production_system/just.html>.

[TPG 2011]TPG. *The Prime Glossary: Sieve of Eratosthenes*. 2011. Disponível em: <http://primes.utm.edu/glossary/xpage/sieveoferatosthenes.html>.

[Valancius et al. 2009]VALANCIUS, V. et al. Greening the internet with nano data centers. In: *Proceedings of the 5th international conference on Emerging networking experiments*

and technologies. New York, NY, USA: ACM, 2009. (CoNEXT '09), p. 37–48. ISBN 978-1-60558-636-6. Disponível em: <http://doi.acm.org/10.1145/1658939.1658944>.

[Varga e Hornig 2008]VARGA, A.; HORNIG, R. An overview of the omnet++ simulation environment. In: *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*. Brussels, Belgium: ICST, 2008. (Simutools '08), p. 60:1–60:10. ISBN 978-963-9799-20-2. Disponível em: <http://dl.acm.org/citation.cfm?id=1416222.1416290>.

[Walker 2008]WALKER, E. Benchmarking Amazon EC2 for high-performance scientific computing. *LOGIN*, v. 33, n. 5, p. 18–23, out. 2008.

[Wang et al. 2010]WANG, L. et al. Cloud computing: a perspective study. *New Generation Computing*, Ohmsha, Ltd., v. 28, n. 2, p. 137–146, 2010. Disponível em: <http://www.springerlink.com/index/10.1007/s00354-008-0081-5>.

[Warneke e Kao 2009]WARNEKE, D.; KAO, O. Nephele: efficient parallel data processing in the cloud. In: *2nd Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS '09)*. Portland, Oregon: ACM, New York, NY, 2009. p. 16–16.

[Wiegand et al. 2003]WIEGAND, T. et al. Overview of the h. 264/avc video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, Citeseer, v. 13, n. 7, p. 560 – 576, 2003.

[Wikipedia 2011]Wikipedia. *List of most watched television broadcasts*. 2011. Disponível em: <http://en.wikipedia.org/wiki/List_of_most_watched_television_broadcasts>.

[wiseGEEK 2012]wiseGEEK. *Clear answers for common questions: What Is Granularity?* 2012. Disponível em: <http://www.wisegeek.com/what-is-granularity.htm>.