Universidade Federal da Paraíba
Centro de Informática
Programa de Pós-graduação em Informática

Coordenação de Infraestruturas Computacionais Distribuídas em Larga Escala Baseadas em Sistemas de Televisão Digital

Diénert de Alencar Vieira

João Pessoa-PB
Janeiro/2013

Universidade Federal da Paraíba
Centro de Informática
Programa de Pós-graduação em Informática

Coordenação de Infraestruturas Computacionais Distribuídas em Larga Escala Baseadas em Sistemas de Televisão Digital

Por

Diénert de Alencar Vieira

Dissertação

João Pessoa-PB
Janeiro/2013

Diénert de Alencar Vieira

Coordenação de Infraestruturas Computacionais Distribuídas em Larga Escala Baseadas em Sistemas de Televisão Digital

Dissertação apresentada ao Programa de Pós-graduação em Informática do Centro de Informática da Universidade Federal da Paraíba, como requisito parcial para obtenção do título de Mestre em Informática (sistemas de computação).

Orientador: Prof.º Dr. Dênio Mariz Timóteo de Sousa
Co-Orientador: Dr. Rostand Edson Oliveira Costa

João Pessoa-PB
2013

Dedico este trabalho aos meus pais que são a minha fortaleza,
a minha namorada que é a minha motivação,
a minha irmã que me coloca nos trilhos
e aos meus amigos que compartilham grandes momentos.

Agradecimentos

Agradeço a Deus por todas as conquistas, oportunidades, aprendizado, saúde e pessoas que colocou no meu caminho que ajudaram a tornar possível todos os sonhos que almejei, consegui realizar e ainda estou em busca da realização. Pelo encaixe perfeito de todas as suas providências, reconheço que eu não poderia ter desejado que acontecesse de forma melhor.

À minha família que está sempre comigo e a quem sempre recorro em momentos de tristeza e alegria, são sempre a minha fortaleza. À minha mãe, Josirene, meu exemplo de carinho, paciência e amor incondicional. Ao meu pai, Vieira, meu exemplo de honestidade, perseverança, humildade e simplicidade. À minha irmã, Daniele, meu exemplo de batalhadora, personalidade forte e determinação.

Um agradecimento especial à minha namorada, Thalita, minha parceira de todos os momentos, quem mais me conhece e me entende. Sem você toda essa jornada não teria sido a mesma coisa, você facilita muito a minha vida. Minha inspiração, meu exemplo para querer ser alguém sempre melhor.

Um agradecimento aos meus amigos de trabalho Diego Pessoa, Sales, Juliana, Ygor, Rafael, Thiago, Alexander, Aécio e Fish, pelos momentos de descontração e risos, pelos momentos de aprendizado, pelos momentos de responsabilidade, pelo nosso saudoso ambiente de trabalho.

Agradeço ao meu orientador, chefe, professor e amigo Denio Mariz por sua atenção e por seu precioso tempo me proporcionando sempre os melhores conselhos e suas explicações com a melhor didática que já tive o prazer de assistir. Assim como agradeço ao meu co-orientador, Rostand Costa, exemplo de profissional, pesquisador, programador, eficiência e cultura. Um dia quero ser como vocês.

Ao professor Guido Lemos pela confiança depositada, pelas horas trabalhadas no LAVID, pela disponibilização de recursos.

Ao professor Francisco Brasileiro (Fubica), por ter aceitado fazer parte da banca e por suas contribuições no grupo de pesquisa que foram de grande valia.

Sem todos vocês, nada disso teria sido possível. Meu muito obrigado!

Resumo

A Infraestrutura Computacional Distribuída sob Demanda (*On-Demand Distributed Computing Infrastructure* – OddCI) utiliza como base uma rede de comunicação em *broadcast* para alocar um conjunto de processadores em larga escala visando à computação de alta vazão (*High Throughput Computing* – HTC). Um exemplo de rede de *broadcast* é o sistema de Televisão Digital cujo sinal é transmitido para milhares de receptores simultaneamente. Esses receptores são máquinas com significativo poder de processamento que estão disponíveis nas residências em quantidade crescente e podem ser utilizados como unidades de processamento. Entretanto, esses potenciais processadores não são completamente dedicados, podem ser cedidos voluntariamente e podem falhar (quando desligados durante o uso), o que os torna recursos altamente voláteis. Em outras palavras, não há garantias sobre o tempo que permanecem dedicados a uma tarefa. Assim, se faz necessária a utilização de mecanismos que tratem essa volatilidade e otimizem a disponibilidade coletiva dos dispositivos. Neste trabalho são investigadas heurísticas para coordenação de sistemas OddCI, que buscam alocar ou liberar dispositivos através da transmissão em *broadcast* de mensagens coletivas com o objetivo de coordenar a quantidade alocada de processadores. Com o propósito de atender acordos de nível de serviço (*Service-Level Agreement* – SLA) estabelecidos, são considerados entre outros fatores a população de dispositivos, sua volatilidade e a quantidade de requisições simultâneas. A eficiência das heurísticas de coordenação foi estudada no âmbito de uma rede de TV Digital, por meio de experimentos de situações extremas com uso de simulação. Como resultados, pudemos identificar os fatores mais significativos, seus efeitos e as restrições resultantes nos cenários avaliados. Em cenários onde a infraestrutura de retaguarda tem capacidade limitada, os principais fatores foram o tamanho da imagem da aplicação utilizada pelas instâncias e a quantidade de instâncias simultâneas, sendo possível atender no caso mais extremo de aplicações de 4MB, 80 instâncias simultâneas e volatilidades de 40% com 50% do paralelismo solicitado. Enquanto em cenários onde *tempo mínimo de finalização* das tarefas era o alvo, os principais fatores foram a volatilidade, a população e, consequentemente, a sua disponibilidade, sendo possível atender 50 instâncias simultâneas com redução de apenas 15% da vazão média solicitada no caso de menor população para volatilidades de até 40%, mostrando até onde os resultados foram favoráveis em cada cenário.

Palavras-chave: computação em grade, computação de alta vazão, sistemas distribuídos.

Abstract

The *On-Demand Distributed Computing Infrastructure* (OddCI) makes use of a broadcast communication network for allocating a large-scale set of processors aimed at High Throughput Computing (HTC). A broadcast network example is a Digital TV system whose signal is transmitted to thousands of receivers simultaneously. These receivers are machines with significant processing power available in our houses in increasing quantities and can be used as processing units. However, these potential processors are not completely dedicated, should be voluntarily ceded and may fail (turned off during the use), which makes them highly volatile resources. In other words, there are no guarantees about the time they remain dedicated to a task. Thus, it is necessary to use mechanisms able to deal with this volatility and to optimize the collective availability of these devices. This work aims at investigating OddCI architecture coordination heuristics, seeking for intelligent ways to allocate or release devices under the coverage of the broadcast network through sending collective messages with the goal of coordinating the allocated processors amount. In order to meet the established Service-Level Agreement (SLA), factors as resources population, volatility and number of simultaneous requests are considered among others. The efficiency of the coordination heuristics has been studied in a Digital TV network environment, through experiments with simulation. As results, we identified the most significant factors, the resulting effects and restrictions in the evaluated scenarios. In the scenario where the backend infrastructure has limited capacity, the main factors were the size of the application image used by the instances and the number of concurrent instances, meeting the most extreme case of 4MB applications, 80 concurrent instances and volatilities of 40% with 50% of the required parallelism. In the scenario where the minimum makespan was the goal, the main factors were the volatility, the population (and the devices availability), and meeting 50 concurrent instances with reduction of only 15% of the required average flow in the case of the smaller population with higher volatilities of up to 40%, showing how far the results have been favorable in each scenario.

Key-words: grid computing, high throughput computing, distributed systems.

Sumário

Lista de Ilustrações

Lista de Algoritmos

Lista de Tabelas

Capítulo 1. Introdução

1.1. Contextualização

A *Lei de Moore* rege a evolução da computação com a predição da duplicação do número de transistores nos processadores a cada 18 meses. A consequência direta é o aumento significativo do poder de processamento nas últimas décadas. Contudo, a taxa de crescimento da velocidade média das CPUs tem se estabilizado, como mostra a Figura 1 nos últimos 5 anos (curva azul escuro). Essa estagnação se deve principalmente pelo custo da energia consumida e pelo calor dissipado. Segundo Ross (2008), a Intel relata que reduzir a frequência de clock de um processador com um único núcleo em 20% economiza metade do consumo de energia, sacrificando apenas 13% do desempenho. Com isso, se o trabalho for dividido em dois núcleos a 80% da taxa de clock, o resultado é um desempenho 73% melhor com o mesmo consumo de energia. Por isso, as empresas fabricantes estão recorrendo ao processamento paralelo com os processadores multicore.

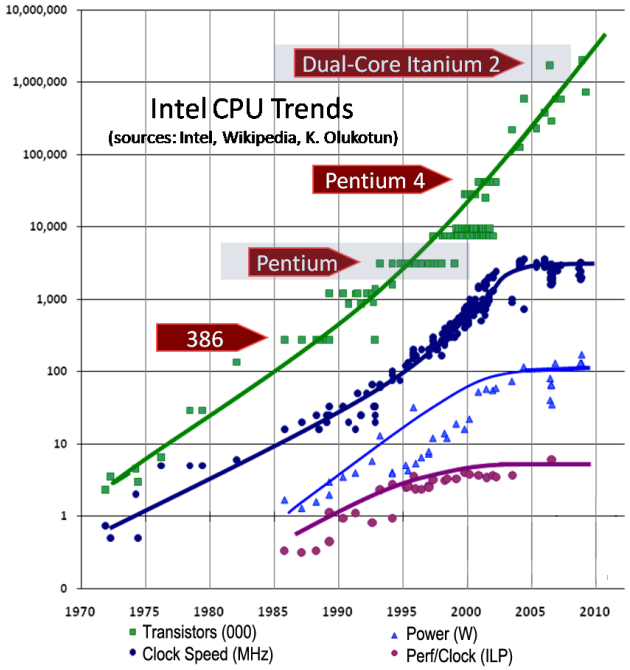


Figura 1 – Número de transistores, frequência de Clock de CPU, consumo de energia e paralelismo a nível de instrução em relação ao tempo (Sutter, 2005)

O processamento paralelo, de longa data, tem provado ser uma abordagem determinante para o futuro da computação. Além do uso em múltiplos núcleos, supercom-

putadores e tecnologias como clusterização, desenvolvida pela IBM na década de 1960, e técnicas de Computação em *Grid* levam a Computação Paralela a paradigmas como Computação de Alto Desempenho (*High Performance Computing* – HPC) (CSTB, et al., 1989), Computação de Alta Vazão (*High Throughput Computing* – HTC) (Litzkow, et al., 1988) e uma combinação das duas anteriores, a *Many Task Computing* – MTC (Raicu, et al., 2008).

Enquanto o paradigma HPC se caracteriza pela necessidade de um grande poder computacional por curtos períodos e se preocupa com a velocidade em que uma tarefa individual executa, o paradigma HTC utiliza esse poder computacional por períodos longos como meses ou anos e se preocupa com a quantidade de trabalhos que são concluídos nesses períodos. Além disso, as tarefas em paralelo na abordagem HPC são fortemente acopladas e na HTC as tarefas são independentes e para fazer a ligação entre as duas abordagens, sistemas MTC se preocupam com a alta vazão de tarefas como o paradigma HTC, mas por curtos períodos e tanto com tarefas dependentes quanto independentes.

Este trabalho tem como foco aplicações do tipo *Bag-of-Task* (BoT) ou *Embarrassingly Parallel Applications* que são aplicações paralelas cujas tarefas são completamente independentes umas das outras e por essa razão são propícias para *grids* computacionais amplamente distribuídos (Cirne, et al., 2003). Este tipo de aplicação pode ser executado em qualquer um dos três paradigmas supracitados, considerando que os recursos de troca de mensagens entre processadores nos paradigmas HPC e MTC estão disponíveis, mas não são utilizados.

Os *Desktop Grids* estão provados como um ambiente HTC adequado, com o sistema Condor (Litzkow, et al., 1988), um dos mais conhecidos representantes da tecnologia, apesar de serem sistemas de escala limitada. Mesmo se mecanismos de incentivo forem usados (Andrade, et al., 2005), é improvável que um sistema com mais que poucas dúzias de milhares de computadores seja montado. De fato, os maiores sistemas existentes usando essa tecnologia se caracterizam por menos que poucos milhares de computadores (Thain, et al., 2006).

Plataformas de computação voluntária, como BOINC (Anderson, 2004) e SETI@home (Anderson, et al., 2002), por outro lado, são aptas a montar grandes amontoados de recursos para processar *workloads* extremamente grandes de suas aplicações típicas. Essas infraestruturas poderosas são, todavia, menos flexíveis no tipo de aplicações suportadas. Primeiramente, configurar uma infraestrutura de computação voluntá-

ria tem um custo que é significativamente maior do que o associado a *Desktop Grids*, devido principalmente ao esforço substancial em convencer voluntários a participar.

A Computação Distribuída trata da quebra de grandes problemas em problemas menores que são executados paralelamente em muitos computadores de propriedade do público em geral, aproveitando o tempo em que seus donos não estão utilizando-os (Godfrey, 2002). Para dar suporte a este tipo de computação existem as Infraestruturas de Computação Distribuída (*Distributed Computing Infrastructures* – DCIs), escalonando, distribuindo tarefas, recolhendo resultados, entre outros procedimentos, em arquiteturas que envolvem uma forma de coordenação desse conjunto de clientes processadores.

O BOINC é um exemplo de DCI. Trata-se de uma infraestrutura aberta que facilita a criação e operação de projetos de computação que utilizam recursos públicos, oferecendo suporte a diversos tipos de aplicações como as que requerem amplo armazenamento e comunicação. Projetos como o SETI@home e o Folding@home (Pande lab, 2011) também utilizam a capacidade ociosa de seus recursos. O primeiro busca por inteligência extraterrestre e o segundo busca pela causa de doenças como Hunntington, Alzheimer, Câncer e BSE (doença da Vaca Louca) através da análise de dobramentos incorretos de proteínas que podem resultar nessas doenças.

OS Type	Native TFLOPS*	x86 TFLOPS*	Active CPUs	Total CPUs
Windows	186	186	178711	4284133
Mac OS X/PowerPC	2	2	2552	152287
Mac OS X/Intel	38	38	9332	174104
Linux	68	68	25204	712414
ATI GPU	1887	1991	13288	332071
NVIDIA GPU	984	2076	6187	278420
PLAYSTATION®3	456	962	16153	1254921
Total	3626	5328	255444	8318887

Figura 2 – Estatísticas dos clientes voluntários do projeto Folding@home (Pande Lab, 2012)

Aproveitar a capacidade ociosa dos computadores pessoais é a forma mais tradicional de montar essas DCIs, conforme o projeto SETI@home que durante 12 meses atingiu uma vazão média de 27,36 TFLOPS. Todavia existem outros dispositivos que permanecem ociosos a maior parte do tempo e possuem significativo poder de processamento. Os consoles de vídeo games são um bom exemplo, entre eles o Playstation 3 que constituiu DCIs de projetos como o Folding@home. Nesse projeto, enquanto máquinas com sistema operacional Windows chegavam a um valor real de 186 TFLOPS,

ver Figura 2, com 178.711 CPUs ativas (esses valores não são os picos de processamento, são as médias), 456 TFLOPS nativos da arquitetura do Playstation 3 são atingidos com apenas 16.153 consoles ativos, o que equivale a 962 TFLOPS na arquitetura x86 (Pande Lab, 2012), apesar da Sony ter finalizado a aplicação de suporte do Playstation 3 para o Folding@home em 6 de novembro de 2012.

Outro tipo de dispositivo que possui capacidade de processamento ociosa em tempo considerável são os receptores de TV Digital. Em 2007, surgiu a primeira arquitetura que visava organizar uma infraestrutura de *grid* computacional utilizando esses receptores, denominada TVGrid (Batista, et al., 2007). Basicamente essa arquitetura usava os canais de *broadcast* (infraestruturas de telecomunicação do tipo um para muitos) em massa e os canais de retorno (um meio de interação como a Internet), como uma conexão de banda larga, para distribuir e coordenar a execução de tarefas paralelas de aplicações BoT. Essa ideia teve continuidade e foi generalizada para tirar proveito de outras infraestruturas que utilizam como base uma rede de comunicação em *broadcast* e possibilitar a alocação de processadores em larga escala, entre elas as redes de telefonia móvel e os sistemas de TV Digital. Essa nova arquitetura é chamada de Infraestrutura Computacional Distribuída sob Demanda (*On-Demand Distributed Computing Infrastructure* – OddCI) (Costa, et al., 2009). Tal arquitetura já possui uma implementação para TV Digital chamada OddCI-Ginga (Costa, et al., 2012c). Visando estabelecer uma prova de conceito, foi feito um *testbed* real com um sistema completo de transmissão e recepção de TV Digital com o padrão do Sistema Brasileiro de TV Digital (SBTVD): gerador de carrossel, multiplexador, modulador, transmissor de baixa potência e alguns receptores TVD com uma versão do *Middleware* Ginga.

Fazendo o paralelo entre a quantidade de dispositivos que o projeto Folding@home atingiu no total, 8.318.887, contando todos os tipos de arquiteturas. Em contrapartida, destaca-se a escala de receptores de TV Digital que pode atingir 38 e 42 pontos no Ibope em novelas do horário nobre em cidades como São Paulo e Rio de Janeiro respectivamente. O que equivale, apenas em São Paulo, com 60 mil domicílios por ponto a 2.280.000 de receptores (UOL, 2013). Unindo mais algumas cidades, pode ser alcançada uma escala sem precedentes.

1.2. Problema e Motivação

Aplicações que executam em receptores de TV Digital estão sujeitas a serem interrompidas a qualquer momento, assim como na computação voluntária. Por não serem recursos dedicados, receptores possuem alta volatilidade, seja por mudança do canal em que a aplicação esteja sendo transmitida ou porque o equipamento pode ser desligado por seu usuário a qualquer momento. Com isso, a tarefa em execução é interrompida e deve ser retomada por outro receptor ou pelo que falhou em outro momento.

Conforme a Norma ABNT NBR 15606-3 (ABNT, 2009b) cada canal (serviço) é associado a uma Tabela de Informações de Aplicação (AIT). A AIT define quais são as aplicações que estão associadas a cada canal. Com a mudança de um canal, uma nova leitura da AIT é realizada, como ocorre a mudança da tabela, as aplicações em execução são finalizadas, mesmo com a utilização da flag UNBOUND, a qual resulta na pergunta se o usuário deseja adicionar a aplicação na lista de aplicações presentes no receptor para execução posterior, mas sempre no canal original da aplicação.

Este trabalho foca no problema de coordenar a alocação desses recursos de forma que seja possível garantir a disponibilidade coletiva definida por Andrzejak (2008), que mostrou que com a utilização de métodos preditivos adequados, 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 ΔT com sobrecarga de controle razoável. Desta forma a DCI pode atender a acordos de nível de serviço (*Service-Level Agreement* – SLA) preestabelecidos, mesmo em cenário onde seus componentes não sejam dedicados e possuam alta volatilidade. Uma descrição mais formal do problema é apresentada no Capítulo 4.

1.3. Questão de Pesquisa e Objetivos

A questão de pesquisa que se pretende responder com este trabalho é: *Existe uma forma de alocar e manter uma infraestrutura confiável de processamento paralelo coordenando receptores de TV digital, não dedicados e de alta volatilidade utilizando um canal de* broadcast *e canais de retorno como a Internet?*

A palavra *confiável* da pergunta se refere à infraestrutura que é capaz de obedecer SLAs e de recusar a aceitar novas requisições quando não puder atendê-las.

Para responder a essa pergunta, foi estabelecido o objetivo principal desta dissertação que é a definição, implementação e avaliação de heurísticas compensatórias para a

volatilidade dos recursos que compõem os sistemas OddCI. E para atingir esse objetivo, foram definidos os seguintes objetivos específicos:

- Definir heurísticas em função do comportamento da volatilidade
- Implementar as heurísticas e validá-las com parâmetros extraídos de um *testbed* real.

1.4. Metodologia

Considerando a abrangência que um sistema OddCI pode alcançar, colocá-lo em produção em uma sistema de TV Digital sem antes testá-lo e validá-lo pode ser muito dispendioso. O esforço necessário para fazer campanhas e definir modelos de negócio que estimulem o usuário proprietário de um receptor a participar da infraestrutura pode ter um custo muito alto, principalmente se for apenas para testes. Por isso, para responder a perguntas iniciais sobre o funcionamento dessa estrutura, foi implementado um primeiro protótipo funcional em pequena escala dessa arquitetura denominado OddCI-Ginga. Esta implementação mostrou a viabilidade tecnológica da proposta, porém ainda é necessário entender a dinâmica da infraestrutura em uma escala extremamente alta como a proposta na definição da arquitetura.

Portanto, para responder a perguntas como "Quantas solicitações de processamento de aplicações BoT são possíveis de ser atendidas em uma cidade com 1 milhão de habitantes em um dia?", a escolha mais factível é a modelagem e simulação do funcionamento da infraestrutura de TV Digital, do OddCI, seus componentes, mensagens e heurísticas. A avaliação descrita neste trabalho foi feita através de simulação, um projeto de experimentos do tipo 2^k fatorial (Jain, 1991) e uma varredura de parâmetros.

1.5. Estrutura da Dissertação

O restante desta dissertação está estruturado da seguinte forma: no Capítulo 2 **– Fundamentação Teórica** são explicados os principais conceitos utilizados para a concepção da arquitetura OddCI, como esses conceitos são aplicados e como é o funcionamento da própria arquitetura. No Capítulo 3 – **Modelo e Implementação do** Sistema OddCI-Ginga – é apresentada a implementação do sistema proposto e sua organização no simulador Omnet++. O Capítulo 4 **– Heurísticas para Coordenação de Infraestruturas Computacionais via Broadcast –** mostra detalhadamente as heurísticas propostas neste

trabalho, que são o núcleo da pesquisa realizada. O Capítulo 5 **– Avaliação de Desempenho –** apresenta como foram feitos os experimentos de verificação do simulador e das heurísticas com a análise dos resultados. No Capítulo 6 **– Trabalhos Relacionados –** abordagens relacionadas são comparadas ao modelo proposto e, por fim, no Capítulo 7 **– Conclusões –** são apresentadas as conclusões do trabalho, as principais contribuições e possíveis trabalhos futuros.

Capítulo 2. Fundamentação Teórica

A arquitetura OddCI usa técnicas de Computação em *Grid*, além de recursos de Sistemas de Televisão Digital. Este capítulo trata destes assuntos e como essas abordagens são utilizadas na arquitetura OddCI.

2.1. Computação em *Grid*

A Computação em *Grid* surgiu em meados dos anos 90 como resultado do crescente interesse multi-institucional em resolver problemas de larga escala. A ideia consiste em aproveitar os ciclos ociosos de computadores de múltiplos domínios administrativos para prover um poder computacional que um único computador, um supercomputador ou um *cluster* em um único domínio não podem prover (Chetty & Buyya, 2002).

De acordo com Foster (2002), um *grid* computacional é um sistema que coordena recursos que não estão sujeitos a controle centralizado, usa protocolos e interfaces padronizados, abertos e de propósito geral e fornece qualidades de serviço não triviais.

Sadashiv e Kumar (2011) apresentam os desafios da Computação em *Grid*:

- Dinamicidade – os recursos são de propriedade e gerenciados por mais de uma organização, podendo entrar e sair do *grid* a qualquer momento, causando uma sobrecarga ao *grid*;
- Administração – para formar um *pool* de recursos unificados, surge uma grande sobrecarga na administração do sistema juntamente com outro trabalho de manutenção para coordenar políticas de administração locais com as globais;
- Desenvolvimento – problema que diz respeito às formas de escrever software para funcionar em plataformas de computação em *grid*;
- Contabilidade – encontrar formas de dar suporte a diferentes infraestruturas de contabilidade, modelo econômico e modelos de aplicação que lidam bem com tarefas que se comunicam frequentemente e são interdependentes;
- Heterogeneidade – encontrar formas de criar uma ampla área de programação de dados intensiva e um framework de escalonamento em um conjunto heterogêneo de recursos;

- Programação – o baixo acoplamento entre os nós e a natureza distribuída do processamento tornam a programação de aplicações para *grid* mais complexa.

As vantagens deste tipo de abordagem computacional são as seguintes (Chetty & Buyya, 2002):

- Possibilitar compartilhamento de recursos;
- Prover acesso transparente a recursos remotos;
- Permitir agregação sob demanda de recursos em múltiplas localizações;
- Reduzir o tempo de execução de aplicações de processamento de dados de larga escala;
- Prover acesso a bases de dados e softwares remotos;
- Tirar proveito dos fusos horários e diversidade aleatória para disponibilidade de recursos;
- Prover a flexibilidade para atender a demandas de emergência imprevistas alugando recursos externos pelo período necessário em vez de possuí-los.

2.1.1. Grid com Arquitetura em Camadas

Foster et. al. (2001) propõem uma arquitetura para *grids* computacionais composta por 5 camadas: *fábrica*, *conectividade*, *recurso*, *coletividade* e *aplicações*.

A camada *fábrica* é composta pelos recursos computacionais, sistemas de armazenamento, catálogos, recursos de rede, entre outros. É sugerido que, no mínimo, os recursos devam implementar mecanismos de auditoria que permitam a descoberta da sua estrutura, estado e capacidades. A camada de *conectividade* engloba o núcleo da comunicação e mecanismos de autenticação como os padrões de segurança: *single sign on* (SSO), delegação de acesso, integração com soluções locais de segurança e relações de confiança baseadas no usuário.

A camada de *recurso* se preocupa com a individualidade do recurso, quanto à negociação segura, inicialização, monitoramento, controle, contabilidade e pagamento de operações compartilhadas. Enquanto que a camada *coletividade* não está associada a nenhum recurso específico, mas com as interações entre as coleções de recursos. E por fim, a camada de *aplicações* que são construídas com base nas chamadas dos serviços definidos em qualquer camada.

Os autores comparam essa estrutura a uma ampulheta, onde a parte estreita são as camadas de recurso e conectividade, pois seus protocolos devem ser escolhidos de

maneira que capturem o mecanismo fundamental de compartilhamento através de muitos recursos de tipos diferentes.

Chetty e Buyya (2002) propõem camadas semelhantes para *grids* computacionais, como mostra a Figura 3 (lado esquerdo), com o acréscimo da camada de ambientes e ferramentas de programação em *grid* e mapeamento direto para as camadas restantes com as propostas por Foster et. al. (2001) Figura 3 (lado direito).

Figura 3 – Arquitetura em camadas para a Computação em Grid (Chetty & Buyya, 2002) (ESQ) (Foster, et al., 2001) (DIR)

2.1.2. Escalonamento de Tarefas

O objetivo de escalonar tarefas em um conjunto de máquinas, como as que compõem um *grid* computacional, é diminuir o *makespan* total das tarefas. *Makespan* é um conceito geral que representa a diferença de tempo entre o início e o fim do processamento de um conjunto de tarefas, o que implica em um *grid* computacional em executar todas as tarefas da forma mais rápida utilizando as máquinas da melhor forma possível. Esse problema é semelhante ao *job-shop problem*. O *job-shop problem* é uma generalização do problema do Caixeiro Viajante, onde o caixeiro é a única máquina e as tarefas são as cidades que devem ser visitadas, só que nesse problema devem ser usadas todas as N máquinas. É sabido que o problema do Caixeiro Viajante é um problema NP-

Difícil, consequentemente, o problema de escalonamento de tarefas também é. Também está provado que é um problema NP-Completo (Ullman, 1975) (Gonzalez & Sahni, 1978). Este não é o problema que se pretende resolver.

2.2. Arquitetura OddCI

Unindo os conceitos de *grid* computacionais e a infraestrutura de TV Digital, Costa et al (2009) propõem a *On-Demand Distributed Computing Infrastructure* (OddCI). Trata-se de uma arquitetura de instanciação sob demanda de DCIs, pois a princípio essa infraestrutura não está montada a priori, sendo iniciada a partir do processo de inicialização, chamado *wakeup process*, após o qual os recursos passam a operar com o objetivo comum de processar solicitações de clientes.

A arquitetura OddCI é composta por um *Provider*, um ou mais *Controllers*, um *Backend*, e os *Processing Node Agents* (PNA). Além disso, pressupõe-se que os PNA estão acessíveis por uma rede de *broadcast* e estão aptos a se comunicarem com os *Controllers* e o *Backend* através de um canal direto de comunicação ponto-a-ponto full-duplex, no âmbito de TV Digital chamado de canal de retorno, ver Figura 4.

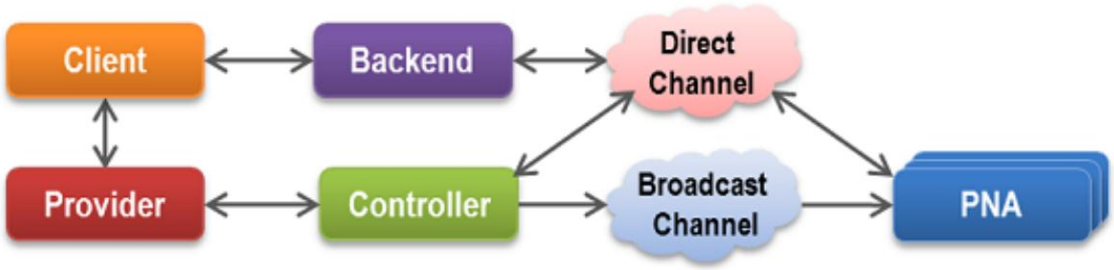


Figura 4 – Visão geral da arquitetura OddCI (Costa, et al., 2009)

A descrição de cada componente é dada na Tabela 1. Além disso, a estrutura de um PNA é formada por três componentes, um *Monitor*, um Ambiente Virtual Dinâmico (*Dynamic Virtual Environment* - DVE) (Keahey, et al., 2004) e a aplicação do usuário, como mostrado na Figura 5. O *Monitor* interage com o *Controller* através do canal de *broadcast*, escutando e processando as mensagens de controle, carregando novas imagens, que são as aplicações que serão executadas pelos recursos no DVE e gerenciando a execução dessas imagens carregadas. Através do canal direto o *Monitor* relata o estado do PNA ao *Controller*. O DVE possibilita um espaço seguro e apropriado para executar a aplicação do usuário (*User Application*), respeitando os interesses do dono do recurso

e do *Client*. E, finalmente, a aplicação do usuário é a imagem carregada que executa o processamento específico desejado.

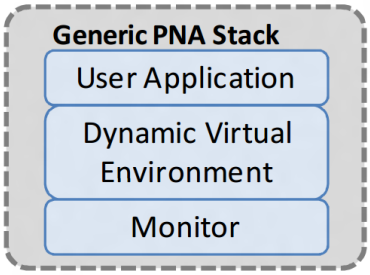


Figura 5 – Pilha do Agente Processador

Tabela 1 – Descrição dos componentes da arquitetura OddCI (Costa, et al., 2012b)

Componente	Descrição
Client	(cliente) é o usuário que deseja rodar aplicações paralelas na infraestrutura OddCI. Ele deve fornecer as aplicações paralelas para o *Provider*, que as repassará para o *Backend*.
Provider	(provedor) é responsável por criar, gerenciar e destruir as instâncias OddCI de acordo com as solicitações dos clientes. O *Provider* é também responsável pela autenticação do cliente e pela verificação das suas credenciais para usar os recursos que estão sendo requisitados.
Controller	(controlador) é encarregado de configurar a infraestrutura, instruído pelo *Provider*. Formata e envia, via canal de broadcast, mensagens de controle e imagens (executáveis) do PNA, necessárias para construir e manter as instâncias OddCI.
Backend	(retaguarda) é responsável pelo gerenciamento das atividades específicas de cada aplicação sendo executada: distribuição de tarefas (aplicações), aprovisionamento de dados de entrada, recepção e 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 comunicação com o *Controller* e *Backend*
Direct Channel	O canal direto é uma rede de comunicação bidirecional que permite a comunicação entre todos os componentes da arquitetura, tal como a Internet.
Broadcast Channel	O canal de broadcast é um canal unidirecional para envio de dados do *Controller* para os dispositivos. Pode ser um canal de TV Digital ou uma ERB de uma rede celular, por exemplo.

2.2.1. **Modelo de operação de um sistema OddCI**

Um *Client* submete uma requisição de instância de DCI para um *Provider*, indicando os requisitos desejados para os recursos, esta requisição é chamada OIR (OddCI *Instantiation Request*), que é representada por uma tupla de 6 elementos:

$$O = < S_S, T_S, P_S, I_A, S_A, C_C >$$

onde S_S é o tamanho médio do *slot* de processamento (unidade de referência de tempo) necessário para executar uma tarefa da aplicação paralela. Este tamanho representa o somatório do tamanho de cada tarefa da aplicação dividido pela quantidade de tarefas.

T_S é a quantidade total de *slots* de processamento necessários para completar a aplicação e representa o produto da quantidade de tarefas da aplicação por S_S. P_S é o nível máximo de paralelismo solicitado ou tamanho da instância, ou seja, o número máximo desejado de dispositivos computacionais ativos na instância em cada momento. I_A é a imagem da aplicação que precisa ser carregada e executada em cada PNA alocado para a instância OddCI. S_A é o acordo de nível de serviço (*Service Level Agreement - SLA*) negociado entre o *Client* e o *Provider* para a instância OddCI em pauta e C_C é a credencial do *Client* que contém as informações necessárias para a autenticação e demais procedimentos de controle de acesso.

O principal componente com o qual um *Client* de um sistema OddCI interage é o *Provider*, que consiste de um conjunto não vazio C_S de *Controllers*, um conjunto H_S de heurísticas de escalonamento de instâncias e um conjunto O_A com todas as OIR ainda ativas:

$$P = < C_S, H_S, O_A >$$

Após a recepção de uma OIR válida, o *Provider* inicialmente decide, considerando as instâncias OddCI que estão correntemente ativas e as características dos seus *Controllers*, se é possível atender aos requisitos contidos na OIR. Se eles podem ser atendidos, o pedido é aceito; caso contrário, o pedido é rejeitado. Se a OIR é aceita, então o *Provider* comanda os *Controllers* mais apropriados para criar a nova instância OddCI.

Uma instância OddCI pode ser alocada pelo *Provider* para um ou mais dos *Controllers* disponíveis dependendo das condições e dos requisitos contidos na OIR. Similarmente, cada *Controller* pode controlar mais de uma instância OddCI. As instâncias OddCI são efetivamente construídas pelos *Controllers* usando os recursos computacionais (ou nós de processamento) que estão conectados através de uma tecnologia de comunicação em *broadcast* específica, capaz de distribuir simultaneamente mensagens para todos os nós conectados.

Um *Controller* individual *C* em C_S é representado aqui por uma rede de *broadcast* B_N, um conjunto H_A de heurísticas de aprovisionamento de instâncias, um conjunto A_I das instâncias OddCI ativas sob o seu controle, um conjunto A_S dos *slots* de processamento alocados para cada instância em A_I, e uma função de custo C_F, a qual é baseada nos requisitos de uma OIR, e retorna o custo unitário para fornecer um slot de tempo nos recursos acessíveis por B_N:

$$C = < B_N, H_A, A_I, A_S, C_F >$$

Por sua vez, uma rede de *broadcast* B_N é representada pela seguinte tupla:

$$B_N = < T_R, D_M, S_D, S_{max}, S_{min} >$$

onde: T_R é a taxa de transmissão do canal de *broadcast*; D_M é o atraso máximo do canal; S_D é o tempo de sessão, que indica o tempo médio estimado que um nó em B_N permanece conectado; e S_{max} e S_{min} são a quantidade estimada máxima e mínima de nós acessíveis por B_N, respectivamente.

Tabela 2 – Símbolos e suas definições

Componente	*Símbolo*	*Definição*
OddCI Instantiation Request (O)	S_S	O tamanho médio do *slot* de processamento necessário para executar uma tarefa da aplicação paralela
	T_S	A quantidade total de *slots* de processamento necessários para completar a aplicação
	P_S	O nível máximo de paralelismo solicitado ou tamanho da instância
	I_A	A imagem da aplicação que precisa ser carregada e executada em cada PNA alocado para a instância OddCI
	S_A	*O SLA* negociado entre o *Client* e o *Provider* para a instância OddCI
	C_C	A credencial do *Client* que contém as informações necessárias para a autenticação e demais procedimentos de controle de acesso
Provider (P)	C_S	Um conjunto não vazio de *Controllers*
	H_S	O conjunto de heurísticas de escalonamento de instâncias
	O_A	O conjunto com todas as OIR ativas:
Controller (C)	B_N	Uma rede de *broadcast*
	H_A	Um conjunto de heurísticas de aprovisionamento de instâncias
	A_I	Um conjunto das instâncias OddCI ativas sob o seu controle
	A_S	Um conjunto dos *slots* de processamento alocados para cada instância em A_I
	C_F	Uma função de custo que é baseada nos requisitos de uma OIR e retorna o custo unitário para fornecer um slot de tempo nos recursos acessíveis por B_N
Rede de *Broadcast* (B_N)	T_R	A taxa de transmissão do canal de *broadcast*
	D_M	O atraso máximo do canal
	S_D	O tempo de sessão
	S_{max}	A quantidade estimada máxima de nós acessíveis por B_N
	S_{min}	A quantidade estimada mínima de nós acessíveis por B_N

2.3. Sistema Brasileiro de Televisão Digital

Conforme Leite e Souza Filho (2005) em suas recomendações para o modelo de referência para o *middleware* procedural do SBTVD, o FlexTV, um sistema básico de transmissão de TV Digital consiste em uma estação transmissora, um meio físico pelo qual o sinal é transmitido e um receptor que recebe o sinal, decodifica-o e o exibe. Para que estas partes se comuniquem, se faz necessário o estabelecimento de um padrão que normatize todo o processo de captura, compressão, modulação e transmissão dos sinais.

Os receptores ou *set-top boxes* são máquinas com processador, memória e sistema operacional como um computador adaptado, embora com um poder computacional inferior, que pode chegar a ser até 27 vezes menor quando comparado a um PC de referência (Costa, et al., 2012c). Estas máquinas executam um *middleware*, que no SBTVD se chama Ginga. O Ginga se divide em duas partes, o Ginga NCL/Lua, declarativo oriundo da proposta MAESTRO (Soares, 2006) e o Ginga-J procedural oriundo do FlexTV (Leite & Souza Filho, 2005). A primeira possibilita o desenvolvimento de aplicações na linguagem de marcação NCL e na linguagem de script Lua, enquanto que o Ginga-J possibilita o desenvolvimento de aplicações em Java.

Apesar desta diferença entre o poder de processamento entre receptores[1] e um PC de referência[2], a aquisição e a evolução do poder de processamento desses receptores é crescente e pode atingir escalas muito altas, como milhões e ainda bilhões de telespectadores (Blog, 2008), o que poderá equivaler a um poder de processamento no nível do paradigma HTC. É importante ressaltar que a abordagem da arquitetura OddCI não foi pensada como um forma de substituir os *grids* que não são formados por receptores de TV Digital, mas como uma alternativa complementar que possui outra forma de processamento não explorada.

O sistema de transmissão de sinal da TV Digital consiste em um mecanismo de comunicação em *broadcast* que independe da quantidade de receptores atingidos. Se um ou um milhão de receptores recebem o sinal, o custo de transmissão é o mesmo. Esta natureza intrínseca do sistema é o cerne da arquitetura OddCI apresentada na Seção 2.2.

[1] Receptores da marca Proview modelo XPS-1000 (firmware 1.6.70, middleware Ginga da RCASoft, 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.

[2] 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.

No SBTVD, um canal de *broadcast* tem uma largura de banda total entre ~18 e ~21 Mbits/s incluindo largura residual para dados entre 1 e 4 Mbits/s, considerando a vazão de um vídeo FullHD codificado em H.264 e uma margem de segurança (ABNT, 2009a) (ABNT, 2009c).

2.4. O comportamento da Audiência Televisiva

Conforme Datia et al. (2005), a importância de sistemas de medição de audiência para o mercado televisivo é, hoje, inegável, pois permite que os canais de mídia se familiarizem com seus consumidores e estejam cientes do seu desempenho em relação a seus competidores. Com isso, destaca-se a importância do comportamento da audiência televisiva para os Sistemas OddCI que são baseados nela.

Um dos principais institutos de medição de audiência no mundo é o instituto Nielsen, líder global de serviços de informação para mídia e indústrias de entretenimento. No Brasil, o principal sistema de medição de audiência pertence ao IBOPE – Instituto Brasileiro de Opinião Pública e Estatística.

Figura 6 – People meter: medidor de audiências do IBOPE

Nas duas empresas, a medição é feita através de um aparelho chamado *People-meter*, apresentado na Figura 6, que é instalado em residências criteriosamente escolhidas para formar a amostragem mais representativa da população. A escolha, no Brasil, é baseada em dados do IBGE. Mas sempre se questiona a confiabilidade dessa medição, que depende da entrada de dados dos telespectadores a cada vez que o aparelho é ligado. Existe um contrato de sigilo para os telespectadores escolhidos, com relação à divulgação da existência do aparelho em suas casas para não haver influência na coleta de da-

dos. Além disso, no Brasil inteiro estão instalados apenas 4000 desses dispositivos sendo que 750 estão na grande São Paulo. (Tecmundo, 2012).

Com base nas medições do Instituto Nielsen, em Hong Kong, Lü (2007) desenvolveu a sua tese "*Modeling of television audience viewing behavior*", propondo modelos probabilísticos de maior aproximação dos valores medidos. Então, para estudar os padrões de difusão de programas de televisão, foi proposto um modelo para medir o primeiro momento de atenção a seriados do horário nobre, utilizando tanto o critério de 1 minuto de permanência quanto o de 15 minutos para considerar que o telespectador assistiu a um determinado programa. A Figura 7a apresenta a visão geral da comparação da medição real, número de telespectadores do modelo proposto, do modelo de Bass e da distribuição Weibull, no critério de 15 minutos do seriado "*War and Beauty*" na cidade de Hong Kong. A Figura 7b mostra a visão de cauda da comparação dos modelos e da medição real do mesmo seriado, enquanto a Figura 7c mostra a visão de cauda da comparação de outro seriado.

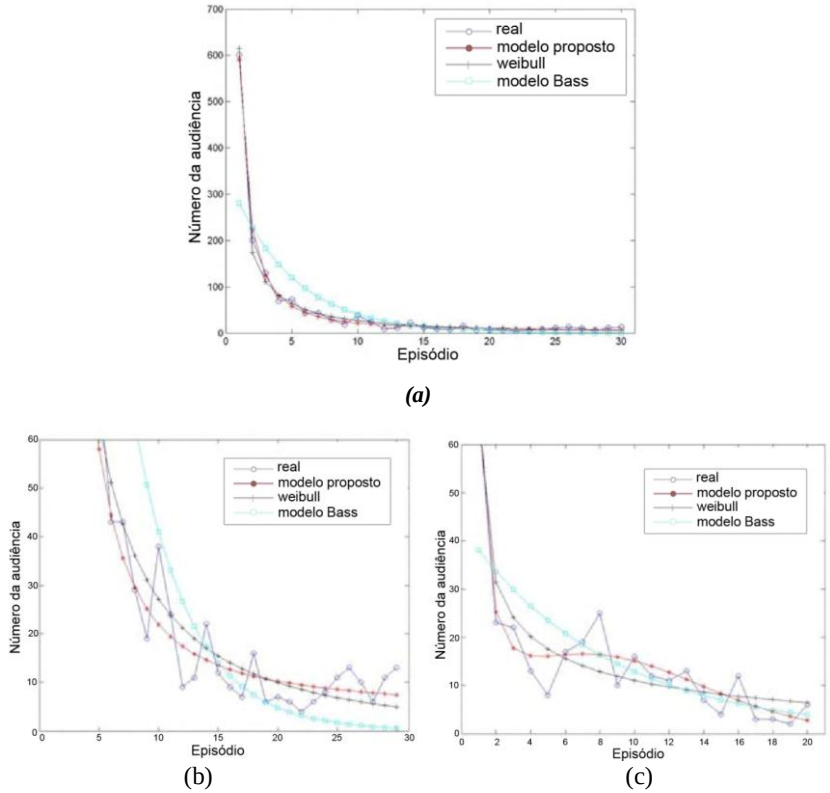


Figura 7 – Comparação da medição real, número de telespectadores do modelo proposto, modelo de Bass e distribuição Weibull no critério de 15 minutos de 2 seriados de canais de Hong Kong a) Visão Geral do seriado A b) visão de cauda do seriado A c) visão da cauda do seriado B

As figuras mostram que a curva (lilás com círculos fechados) do modelo proposto se aproxima mais da medição empírica (curva com mudanças abruptas e círculos

abertos), enquanto que a distribuição Weibull (curva com cruzes) tende a superestimar em um estágio anterior e subestimar em um estágio posterior porque geralmente tem uma cauda mais fina que o modelo proposto.

Até determinado momento deste trabalho foi feita uma modelagem do comportamento da audiência televisiva usando-se uma Distribuição Pareto com:

$$\alpha = \frac{24 - \overline{T}}{24 - \overline{T} - 1}$$

onde 24 é quantidade de horas do dia, $\overline{T}$ é o tempo médio de duração de sessão de um telespectador, α é o parâmetro da forma da distribuição e k=1, o segundo parâmetro da distribuição.

Todavia, este modelo não permitia manipular diferentes situações de audiência como horário nobre. Por razões de facilitação de manuseio da audiência, optou-se por inserir um fator de volatilidade para representar a audiência em cada momento específico. Mais detalhes sobre este processo são apresentados na Seção 3.4 através do algoritmo que determina quando os PNAs são ligados e desligados, o *liveOrDie.*

2.5. Previsão do estado de recursos

Como a seção anterior trata da previsão do comportamento de audiências, também se faz necessário um modelo de previsão do uso dos recursos constituintes dos sistemas voluntários, para isso, alguns trabalhos da literatura são apresentados.

A disponibilidade de recursos no caçador de máquinas ociosas – Condor – é modelada em 5 estados (Litzkow, et al., 1988) (Rood & 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. Alguns desses estados não estão presentes em alguns contextos em que o OddCI pode operar. Por exemplo, o estado usuário presente, que é disparado quando o teclado ou o mouse são tocados, não é um estado válido, dependendo do tipo de recurso utilizado.

Com base nesses estados e no histórico de disponibilidade dos recursos, Rood e Lewis (2009) propõem preditores para análise de intervalos considerando os *N* dias anteriores no mesmo horário da previsão (*N-Day)* e considerando as *N* horas anteriores ao horário da previsão (*N-Recent).* A abordagem *N-Recent* é semelhante à técnica de sele-

ção por ranqueamento proposta neste trabalho, seção 4.2.1. Com relação à técnica de análise, os autores consideram o número de transições do estado disponível para cada outro estado de indisponibilidade *(transitional)*, calculam a porcentagem de tempo que o recurso permanece em cada estado e utilizam este valor como a probabilidade do recurso mudar para o estado a seguir (*durational)*. 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 com peso 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 o maior peso 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 preditores como *Saturating and History Counter predictors* (Mickens & Noble, 2006), *Multi-State and Single State Sliding Window predictors* (Dinda & O'Hallaron, 1999), *Ren predictor* (Ren, et al., 2007) *e Completion predictor* que sempre prevê que um trabalho completará no intervalo requisitado, o resultado pode ser visto na Figura 8.

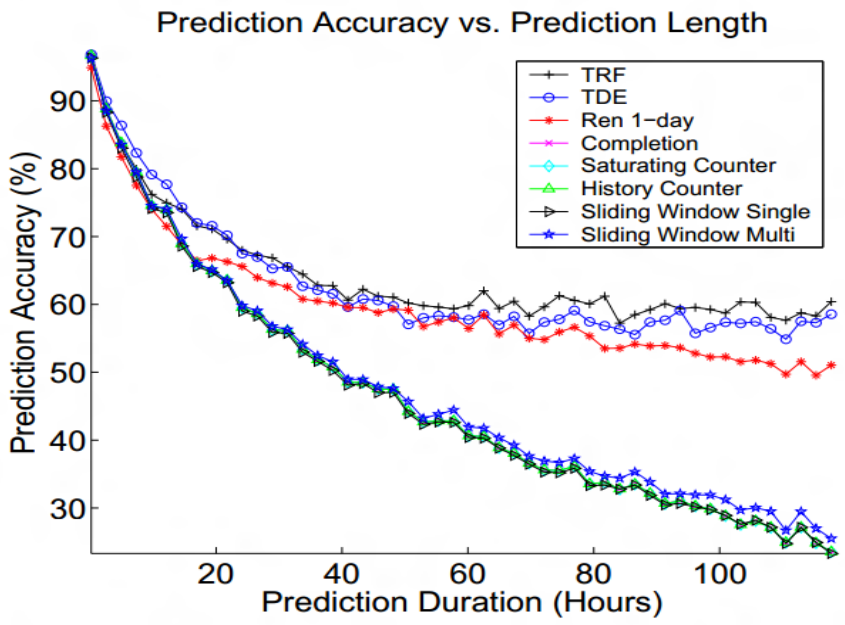
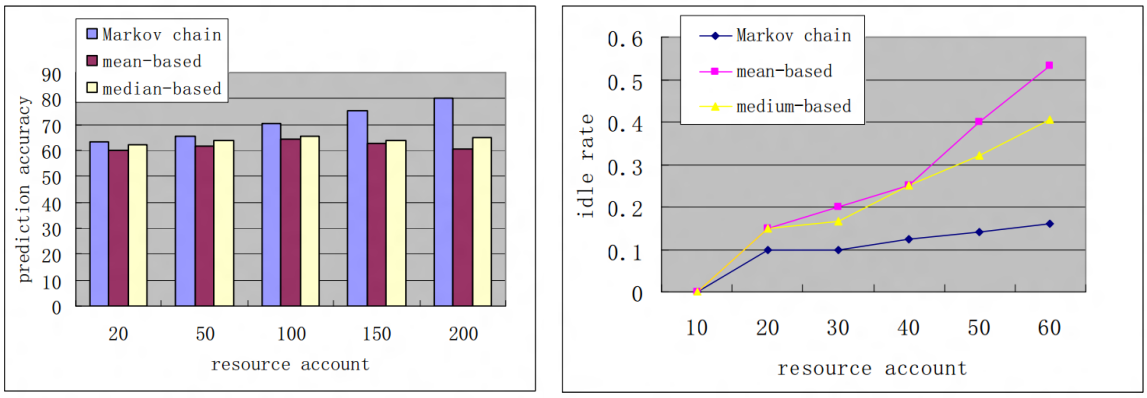


Figura 8 – Comparação de acurácia em relação ao tempo decorrido (Rood & Lewis, 2009)

Esforços de previsão de estados de recursos avaliam históricos de uso da CPU, rede, disco e memória. Um exemplo de abordagem notável para sistema de previsão para *grids* é o *Network Weather Service* (NWS) (Wolski, 1998) (Wolski, et al., 1999)

que, com utilitários UNIX, implementa seus mecanismos de sensores para o hardware do recurso de forma pouco intrusiva, ou seja, o monitoramento impõe pouca sobrecarga ao recurso.

Em comparação com o NWS, considerando o uso da CPU, da rede e a taxa de falha, Lili, *et al.*, (2009) apresentam um método de predição baseado em uma cadeia de Markov utilizando uma matriz com as probabilidades de transições entre os possíveis estados. Como resultado, uma acurácia de predição – descrita no trabalho como o nível de conformidade entre o estado previsto e o estado real – maior que a média e a mediana utilizadas nos métodos de predição do NWS, ver Figura 9a, além de uma taxa menor de recursos ociosos, ver Figura 9b. Isso se deve ao fato de que a média e a mediana não podem refletir mudanças bruscas de estado, não podendo reagir tão rapidamente quanto o método proposto da cadeia de Markov.



(a) (b)

Figura 9 – a) Comparação da taxa de recursos ociosos b) comparação da acurácia de predição (Lili, et al., 2009)

Hanzich, *et al.*, (2011) propõem um sistema de escalonamento de tarefas orientado a ambientes não dedicados, *Cooperative & Integral Scheduler for Non-dedicated Environments* (CISNE) que possui uma abordagem online – cluster real, onde uma nova estimativa de tempo de resposta é feita a cada chegada de trabalho – e uma abordagem off-line – cluster simulado, que estima o comportamento de cargas de trabalho paralelas sobre ambientes de simulação de eventos discretos. Essa segunda abordagem é a que mais se aproxima da proposta deste trabalho. A Figura 10 apresenta o esquema híbrido da abordagem online, a união dos registros históricos para inferir o futuro como uma replicação do passado e um esquema de simulação em tempo discreto baseado em um modelo analítico como mecanismo núcleo de previsão do tempo de resposta dos nós.

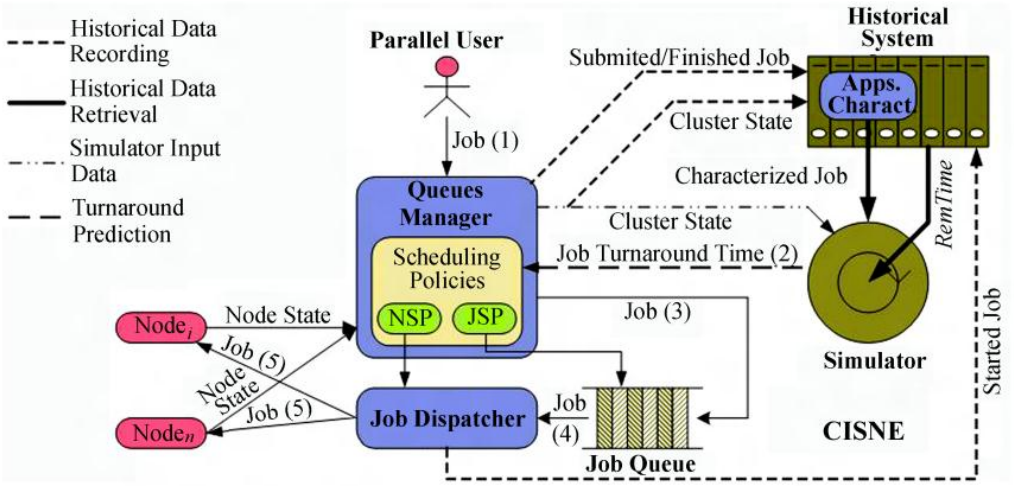


Figura 10 – Abordagem Online – integração de esquema híbrido (Hanzich, et al., 2011)

A Figura 11 apresenta a arquitetura do esquema offline. Onde os nós/recursos fazem parte do mecanismo de simulação, além do próprio núcleo de simulação para previsão do tempo de resposta. A ideia principal é que, através da simulação, seja possível entender não apenas o comportamento de um certo tipo de carga de trabalho paralela, mas muitos tipos de ambientes, variando sua heterogeneidade, tamanho, carga local, recursos computacionais dos nós, entre outros.

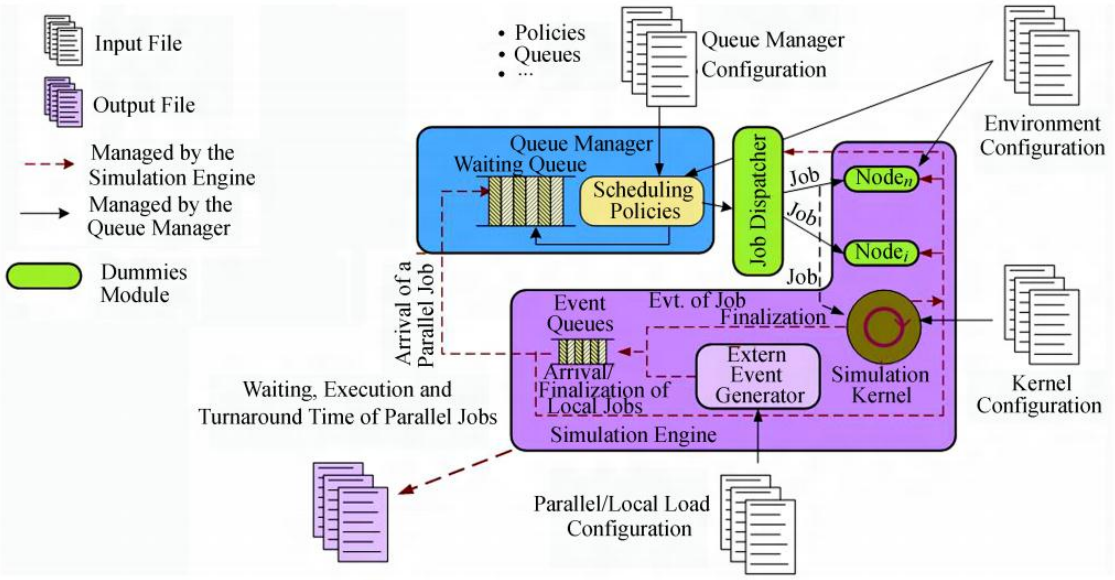


Figura 11 – Arquitetura do esquema off-line (Hanzich, et al., 2011)

Os autores realizaram um monitoramento durante 1 mês de um laboratório de sua universidade, classificando a utilização dos computadores e utilizando estes resultados para criar um *benchmarking* sintético para testar o seu sistema tanto no ambiente online quanto off-line. E concluiu que sua abordagem híbrida, esquema da Figura 10,

teve os melhores resultados por combinar previsão através de simulação juntamente com dados históricos de um cluster real.

A abordagem utilizada neste trabalho é muito semelhante ao esquema off-line proposto, uma vez que os nós fazem parte da simulação assim como o mecanismo de previsão de disponibilidade implementado através das heurísticas de aprovisionamento tratadas na seção 4.3. Esta escolha se deve principalmente ao custo de testes a serem realizados em larga escala, ao sistema estar em fase de validação e à facilidade de configuração de um ambiente de simulação em relação a um ambiente real.

Capítulo 3. Modelo e Implementação do Sistema OddCI-Ginga

Neste capítulo, é apresentada a descrição do comportamento de um sistema OddCI. Este modelo consiste em um algoritmo de manipulação da volatilidade que se deseja inserir no modelo em um dado instante, representando a permanência de telespectadores em um canal, as entidades e mensagens trocadas entre elas, os parâmetros de configuração do ambiente como taxas de transmissão e tamanho das imagens enviadas via broadcast. As heurísticas são discutidas no Capítulo 4.

3.1. Simulador Omnet++

A principal ferramenta utilizada neste trabalho foi o Omnet++ – *Objective Modular Network Testbed – The Open Simulator* (Varga & Hornig, 2008). O Omnet++ é uma biblioteca e framework de simulação em C++, extensível, modular e baseado em componentes, inicialmente para construção de simulações de rede. Rede no sentido amplo, que inclui redes de comunicação com fio, *wireless*, redes em chip (*NoC),* redes de filas, entre outras. O Omnet++ oferece uma IDE baseada no Eclipse, um ambiente gráfico de execução, além de outras ferramentas, extensões e projetos independentes como, por exemplo, suporte a redes de sensores, redes ad-hoc e protocolos da Internet.

A ideia básica desse simulador de eventos discretos é a modelagem de componentes e a sua comunicação através de troca de mensagens que podem representar, por exemplo, frames ou pacotes. Para isso, é utilizada uma linguagem de descrição de topologias chamada NED (*Network Description*), com a qual é possível descrever a estrutura dos módulos e a forma como eles se comunicam entre si. Os módulos compostos são totalmente descritos pela linguagem NED, enquanto que os módulos simples, além da topologia em NED, têm a lógica implementada em C++.

O funcionamento interno de componentes simples no Omnet++ é determinado pelo agendamento de mensagens de controle, denominadas *timers*. Assim, caso o componente tenha que executar alguma tarefa, ele agenda para ele mesmo um *timer* com uma flag sinalizadora da ação a ser tomada e o momento em que esta ação será executada.

Os componentes possuem, por padrão, uma rotina *handleMessage* que trata tanto as mensagens recebidas de outros componentes, como também é responsável por delegar o tratamento dos *timers* para a função *handleTimer*. Uma descrição detalhada das mensagens entre componentes é apresentada na seção seguinte, os *timers* são apresentados nas seções dos seus respectivos componentes.

A Figura 12 apresenta o ambiente gráfico de execução do simulador onde parte da modelagem do sistema OddCI está representada: um *Controller*, grupos de PNA e um componente responsável somente pela transmissão das mensagens de *broadcast*, o *Transmitter* que atua como um *gateway* para a emissora de um canal de TV Digital. Esse ambiente é de grande utilidade para fins de demonstração e didática. O ponto vermelho é uma *wakeup message* sendo enviada do *Controller* para o *Transmitter* e os pontos azuis, são a *wakeup message* sendo enviada em *broadcast* para cada PNA. As linhas dos roteadores para os grupos de PNA são os canais de retorno e o círculo com o *Transmitter* no centro representa o seu alcance de transmissão.

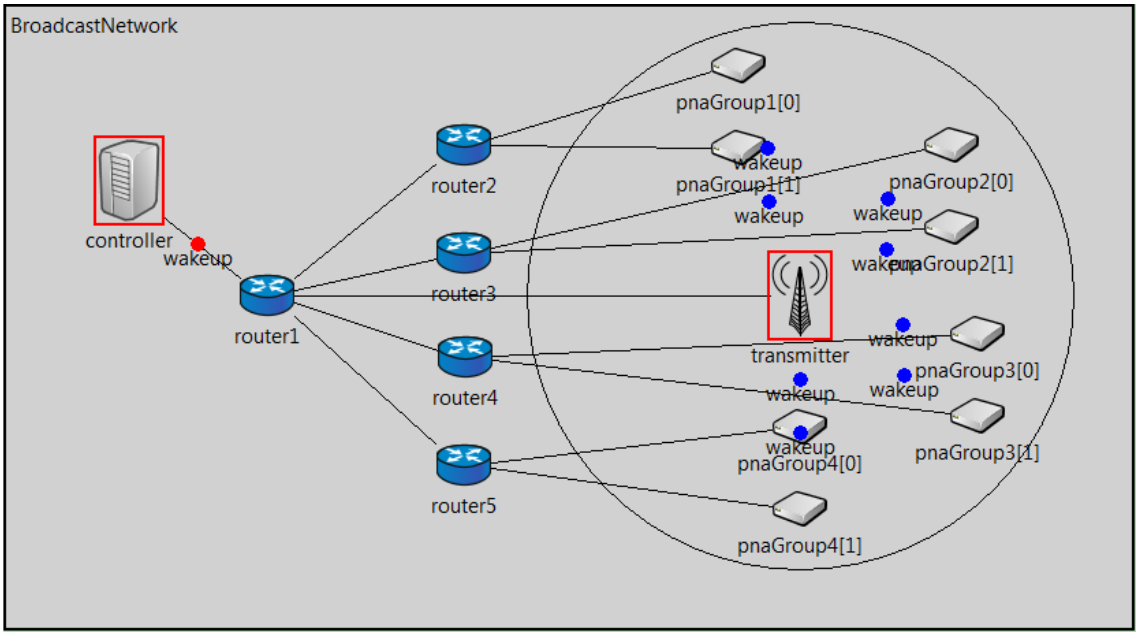


Figura 12 – Ambiente gráfico de execução com simulação do OddCI no momento do broadcast da wakeup message

Na Figura 12, estão presentes apenas 8 PNA, mas os experimentos foram executados com números muito maiores, tornando inviável a execução em interface gráfica. A barra de ferramentas (retângulo laranja na Figura 13) é onde as principais ações estão disponíveis, a barra de status (retângulo verde da Figura 13) apresenta informações do estado corrente da simulação, como o número da execução/repetição, número do evento, tempo da simulação e outras informações. A linha do tempo (retângulo amarelo da Fi-

gura 13) mostra os eventos em escala de tempo logarítmica, incluindo os agendados. A árvore de objetos (retângulo vermelho da Figura 13) exibe todos os objetos inspecionáveis presentes na memória. E por fim, a área de *log* (retângulo azul da Figura 13) apresenta a saída de toda a simulação, que pode ser filtrada por módulos específicos.

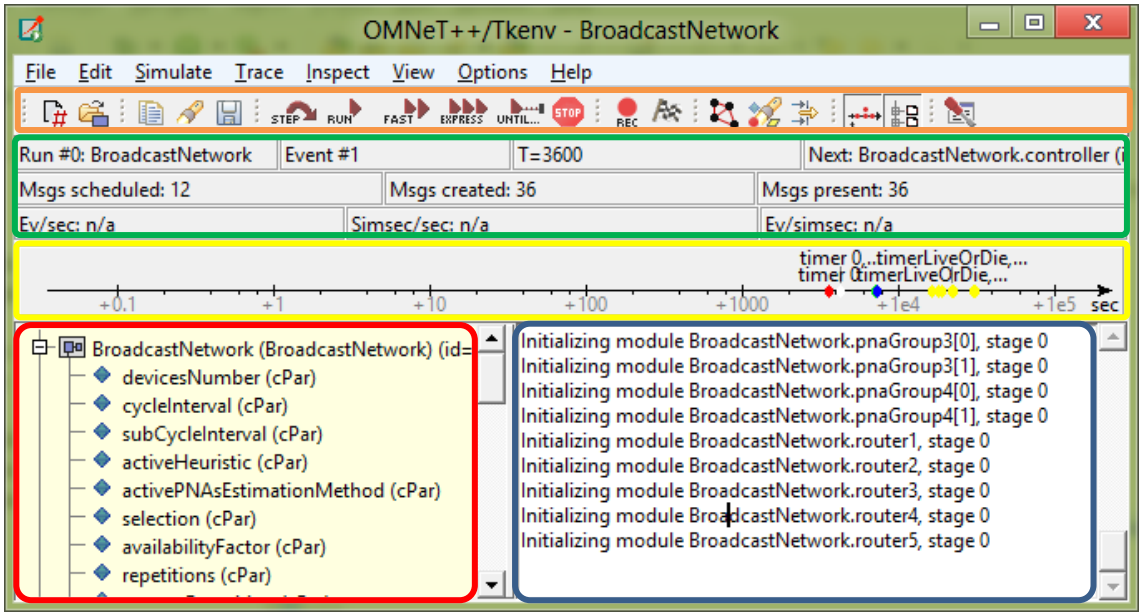


Figura 13 – Tkenv – ambiente de monitoramento e controle de execução

Além do modo gráfico de execução, o simulador dispõe de uma execução em linha de comando, sem animações, apenas com uma saída equivalente à barra de status, que foi o modo mais utilizado nestas simulações. O modo gráfico serviu de auxílio nos momentos de verificação da simulação com número menor de elementos.

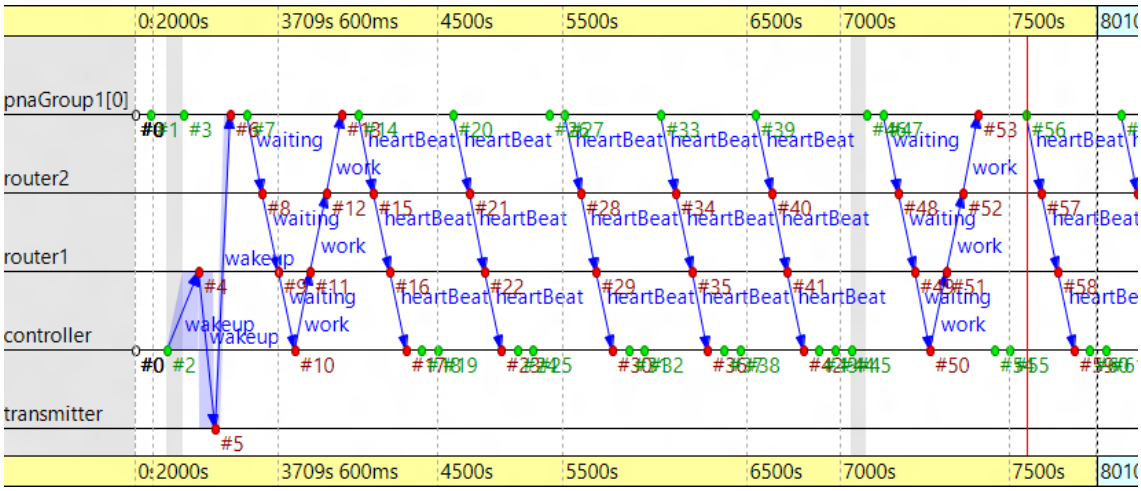


Figura 14 – Diagrama de sequência gerado pelo Omnet++ para o OddCI

Entre os vários recursos do simulador utilizados, um de grande importância é o gerador de gráfico de sequência. Para simplicidade, a Figura 14 exibe a sequência de uma rede OddCI com apenas 1 PNA e 1 única instância ativa. Neste gráfico, é possível

ver o envio de uma *wakeup message*, evento #2, passando pelo *router1*, evento #4, chegando ao *transmitter*, evento #5, até chegar ao PNA, evento #6 e este, por sua vez, responder ao *controller* com uma mensagem do tipo *waiting*, eventos #7 a #10. Está presente na figura o envio de uma mensagem do tipo *work* do *controller* ao PNA, sinalizando a entrada de fato do PNA na instância, do evento #10 ao evento #13. Também se pode observar o envio de sondas sucessivas, *heartBeats*, do PNA ao *Controller* do evento #14 ao evento #42 e a finalização deste processo inicial com uma nova mensagem do tipo *waiting*, do evento #47 ao #49. Vale ressaltar que o canal de *broadcast* só foi utilizado para a difusão da *wakeup message*, além do que já foi citado, as outras mensagens de controle são enviadas através dos canais de retorno.

3.2. Ciclos de processamento e controle temporal

Para controlar os componentes e para o entendimento do restante do documento, alguns termos são utilizados e seus conceitos são explicados a seguir.

Um **slot** é a unidade de tempo de processamento executado por um único dispositivo de forma ininterrupta por um período de duração pré-estabelecida que, ao ser concluído, atende a menor parte do processamento requisitado por uma instância. A finalização com sucesso de uma instância significa que a soma dos slots completados é maior ou igual ao requisitado.

Um **ciclo** para o *Controller* é a unidade de tempo que agrupa a finalização de vários slots, ou seja, dispositivos diferentes em paralelo, cada um responsável por processar um slot. Esse agrupamento leva em consideração o paralelismo representado pela quantidade de dispositivos requisitados pela instância para resultar no processamento do conjunto total de slots. Para um PNA, o término de um ciclo de processamento consiste em completar slot ao qual foi designado.

Um **subciclo** é o intervalo de tempo mínimo no qual uma instância é monitorada. Nos nossos experimentos, que consideram contextos de alta volatilidade, ele é uma fração da duração do ciclo. Normalmente, o HBI (*heartbeat interval*) atribuído pelo *Controller* para o envio de sondas pelos PNAs está coordenado com a duração adotada para o subciclo. Caso a sonda não seja enviada do PNA para o *Controller*, presume-se que o dispositivo falhou e novos dispositivos são alocados para compensá-lo.

O **workload** é descrito para cada instância pelo tempo de início, pela quantidade de dispositivos em paralelo requisitados por ciclo e pelo total de slots que devem ser processados.

3.3. Mensagens

As mensagens trocadas entre as entidades OddCI modeladas possuem o identificador da instância à qual está associada e também o endereço de origem e o endereço de destino da mensagem, utilizados nos canais de retorno. As mensagens WakeupMsg, DieMsg e WorkMsg na Figura 15 são mensagens enviadas do *Controller* para os PNAs e mudam o seu estado corrente, enquanto que as mensagens IdleProbeMsg, HeartBeatMsg e WaitingMsg são mensagens enviadas dos PNAs para o *Controller* e informam o estado corrente do PNA.

Uma *wakeupMsg* inicia o *wakeup process* e é enviada pelo *Controller* ao *Transmitter* e este se encarrega de fazer o *broadcast* para os dispositivos. Ao receber uma *wakeupMsg*, o PNA responde pelo canal direto com uma *waitingMsg*, indicando que está aguardando autorização para começar o processamento ou ser descartado. Ao receber uma *waitingMsg*, o *Controller* decide pelo ingresso do PNA na instância, caso positivo ele envia-lhe uma *workMsg*. Caso decida descartá-lo, ele envia-lhe uma *dieMsg* porque o número de dispositivos alocados já atingiu o necessário.

Uma *workMsg* indica em qual ciclo o dispositivo deve iniciar o seu processamento, qual é o tempo que ele deve permanecer ativo para concluir um slot. Além disso, há o intervalo de tempo em que o dispositivo deve enviar a primeira sonda a partir do recebimento da *workMsg*, a propriedade *firstHeartBeatInterval*, e a partir dessa primeira sonda o intervalo para envio das sondas subsequentes com a propriedade *heartBeatInterval*.

Uma *dieMsg* faz com que o dispositivo que estava em um estado *waiting* ou *busy* volte a ficar disponível podendo ser alocado em qualquer instância, incluindo a própria que o descartou em outro momento.

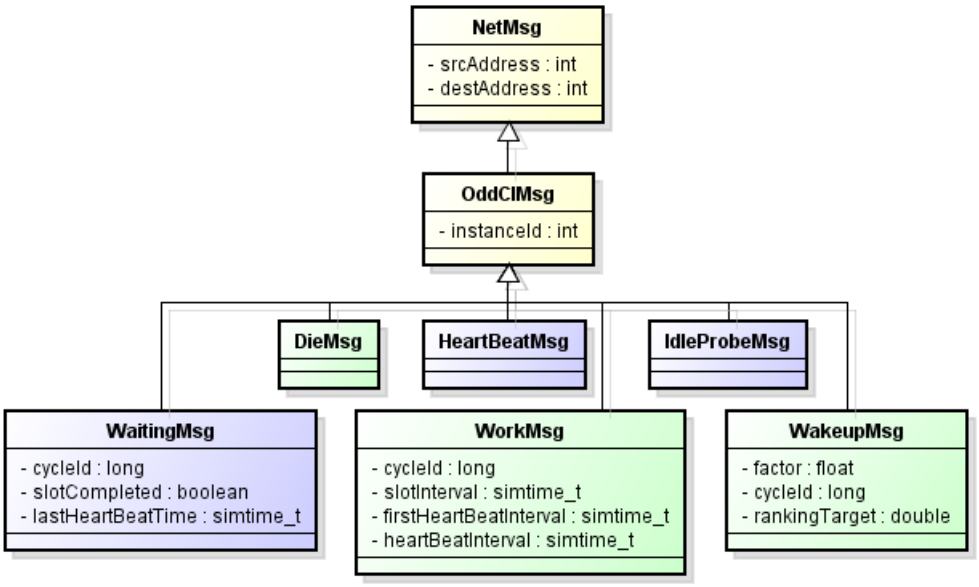


Figura 15 – Mensagens trocadas entre os componentes da arquitetura

Durante o processamento, o PNA envia para o *Controller* sondas regulares para indicar que ainda está ativo, chamadas de *heartBeatMsg* ou sondas de atividade. E para que o *Controller* saiba quantos dispositivos estão ativos em cada momento dentro de um ciclo, existem os subciclos. Os subciclos são períodos em que o *Controller* reinicia a contagem dos dispositivos e a refaz, para mantê-la atualizada. O subciclo permite o acompanhamento do estado da instância de forma mais granular, o que nem sempre é possível dependendo do tamanho da duração do ciclo, o subciclo confere maior flexibilidade ao sistema. Com base principalmente na contagem em cada subciclo, caso não atinja o número necessário, novas *wakeupMsgs* são enviadas dentro de um mesmo ciclo ou até dentro de um mesmo subciclo para alocar mais dispositivos. Este último pode acontecer na heurística de paralelismo limitado explicada na Seção 4.3.

A *idleProbeMsg* é uma sonda especial para os dispositivos que estão disponíveis, ou seja, não estão alocados em nenhuma instância. Este é um dos métodos de estimativa do total de dispositivos disponíveis na rede, chamado **método informativo**, explicado na Seção 4.1.

Ao completar um slot de processamento, no fim de um ciclo, existe uma confirmação para que o PNA continue ou não o processamento. Por isso, o PNA envia novamente uma *waitingMsg* também indicando que aguarda essa confirmação, mas dessa vez com a flag *slotCompleted = true* para diferenciá-lo dos dispositivos que estão iniciando na instância e informando qual ciclo foi finalizado com a propriedade *cycleId*. Dependendo do momento em que os dispositivos recebem uma *wakeupMsg*, a sua *wai-*

tingMsg pode ser usada para contabilizá-lo como trabalhando para a instância, visto que pode não haver tempo suficiente para ele enviar uma *heartBeatMsg* ainda no mesmo subciclo, a propriedade *lastHeartBeatTime* serve para auxiliar neste procedimento, evitando que o mesmo dispositivo seja contado duas vezes como ativo, caso este já tenha enviado uma *heartBeatMsg* no subciclo.

3.4. Controle de Estados e Adesão dos PNAs

As mensagens WakeupMsg, DieMsg e WorkMsg da Figura 15 manipulam em parte os estados dos PNAs que podem ser *idle, busy, waiting, on* e *off* . Os estados *idle*, *busy* e *waiting* compõem o estado *on*, ou seja, ligado. Além das mensagens, uma rotina chamada *liveOrDie*, que decide se o PNA continua ligado (1) ou desligado (0), é executada repetidas vezes com o intervalo de um ciclo começando em um ponto determinado por uma variável aleatória com distribuição uniforme U[0, t], onde t é o tamanho do ciclo. Inicialmente, todos os dispositivos estão no estado *off*, a rotina *liveOrDie* é chamada e caso resulte em ligá-lo, o PNA muda para o estado *idle*, como mostrado na Figura 16. O estado *idle* significa que o dispositivo está sintonizado no canal em que o OddCI está configurado, mas o PNA não está alocado em nenhuma instância e, consequentemente, está disponível.

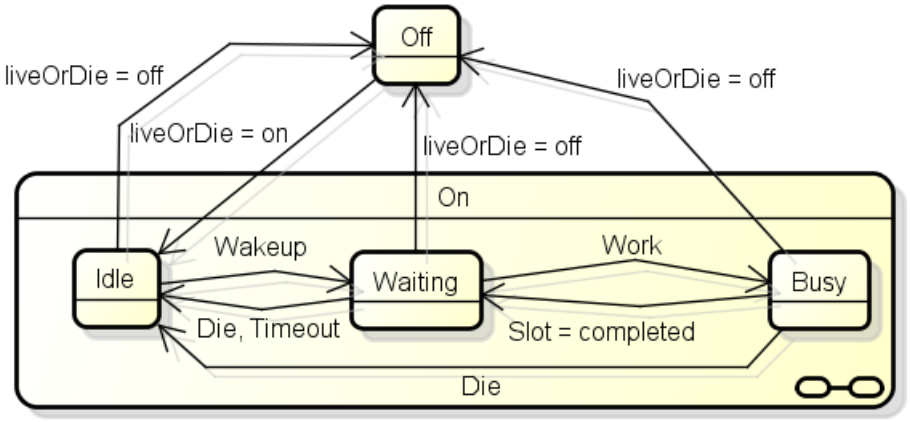


Figura 16 – Diagrama de estados de um PNA

Caso o PNA receba uma *wakeupMsg* (WM), este é um potencial candidato a fazer parte da instância, mudando para o estado *waiting*. Após um *timeout*, caso não tenha recebido uma *dieMsg* ou uma *workMsg*, o PNA volta para o estado *idle* automaticamente para voltar a responder a *wakeupMsgs*. Caso o PNA receba uma *workMsg*, ele muda para o estado *busy* que significa que o dispositivo está alocado em uma instância e está

executando alguma tarefa. Caso receba uma *dieMsg*, ele volta para o estado *idle*. Quando o PNA está nos estados *waiting*, *busy* ou *off*, ele ignora as WM recebidas.

O estado *off* é uma abstração de alguns estados não descritos no diagrama. Esses estados são: o dispositivo está desligado, o dispositivo não está sintonizado no canal em que o OddCI está configurado, o dono do dispositivo não desejou fazer parte do OddCI, ou seja, situações que impossibilitem a participação do dispositivo em instâncias OddCI.

As operações internas dos componentes são gerenciadas principalmente por mensagens agendadas por eles para eles mesmos, denominadas *timers*, como as operações *liveOrDie*, enviar sondas e concluir ciclos. O algoritmo *liveOrDie* é apresentado no Algoritmo 1:

Algoritmo 1 – liveOrDie

```
cycleInterval = Duração do intervalo de um ciclo
firstLiveOrDie = Boleano para a primeira execução da rotina
numDevices = total de dispositivos existentes na rede de Broadcast
initialAvailability = Fator de disponibilidade inicial relativa à população de dispositivos
volatility = porcentagem de volatilidade inserida na simulação
onPNAs = contador de dispositivos no estado ON
idlePNAs = contador de dispositivos no estado IDLE
busyPNAs = contador de dispositivos no estado BUSY
waitingPNAs = contador de dispositivos no estado WAITING

1       nextLiveOrDie = cycleInterval
2       IF IS firstLiveOrDie THEN
3               nextLiveOrDie = uniform(0, cycleInterval)
4               nextState = bernoulli(initialAvailability)
5               firstLiveOrDie = FALSE
6       ELSE
7               IF state = OFF THEN
8                       nextState = bernoulli((numDevices * initialAvailability * volatility) / (numDevic-
        es * (1 - initialAvailability)))
9               ELSE
10                      nextState = bernoulli(1-volatility)
11              ENDIF
12      ENDIF
13      IF nextState = OFF THEN // Desligando um PNA que estava no estado ON
14              IF state <> OFF THEN
15                      onPNAs = onPNAs - 1
16                      IF state = IDLE THEN
17                              idlePNAs = idlePNAs - 1
18                      ELSE IF state = BUSY THEN
19                              busyPNAs = busyPNAs – 1
20                      ELSE IF state = WAITING THEN
21                              waitingPNAs = waitingPNAs – 1
```

```
22                  ENDIF
23                  cancel slotTimer
24                  cancel heartBeatTimer
25                  cancel checkAvailabilityTimer
26          ENDIF
27          state = OFF
28          instanceId = 0
29      ELSE
30          IF state = OFF THEN // Ligando um PNA que estava no estado OFF
31              onPNAs = onPNAs +1
32              idlePNAs = idlePNAs + 1
33              state = IDLE
34              schedule(checkAvailabilityTimer, now+checkInterval)
35              set messageKind in slotTimer for IDLE_PROBE
36              schedule(slotTimer, now)
37          ENDIF
38      ENDIF
39      schedule(liveOrDieTimer, now+nextLiveOrDie)
40      RETURN
```

A variável *initialAvailability*, linhas 4 e 8 é um valor entre 0 e 1 e representa a porcentagem do total de dispositivos na simulação, *numDevices*, que deve ficar disponível aproximadamente. *Bernoulli* é uma função de distribuição de probabilidade que retorna 1 com a probabilidade *initialAvailability* passada como parâmetro e 0 com a probabilidade de 1 - *initialAvailability.*

A linha 8 do algoritmo representa o próximo estado do PNA (0 = OFF e 1 = ON) considerando os parâmetros de disponibilidade inicial (qual a disponibilidade inserida na rede inicialmente, ou seja, uma porcentagem do total de dispositivos que iniciará no estado ON), de volatilidade (qual a taxa de falha inserida no sistema) e que este PNA estava no estado OFF. A linha 10 é o próximo estado do PNA, mas considerando que este PNA estava no estado ON. A volatilidade inserida na simulação é controlada por estas duas linhas, que foram planejadas de forma que a quantidade de dispositivos que ficam OFF em um momento, seja reposta na mesma quantidade por outros PNAs.

Da linha 13 à 28, está sendo tratado o caso do PNA estar ligado e vai passar a estar desligado, efetuando as contagens específicas e cancelando os *timers* que só devem funcionar quando o PNA está em atividade, além de atualizar o estado para OFF. Da linha 29 à linha 38 está sendo tratado o caso em que o PNA estava desligado e vai passar a estar ligado, agendando novamente os *timers* de checagem de disponibilidade (linha 34), e o imediato envio de sonda *idleProbeMsg* indicando a disponibilidade ao *Con-*

troller, linhas 35 e 36. Por fim, na linha 39, o agendamento da próxima chamada da própria rotina *liveOrDie*.

A Figura 17 apresenta o diagrama de sequência de um ciclo de processamento envolvendo os 3 principais componentes da arquitetura OddCI. A mensagem 1 representa a rotina *liveOrDie*, a mensagem 2, a sonda de disponibilidade, a mensagem 3 e as suas subsequentes constituem o processo de *wakeup* difundido pelo *Transmitter*. A mensagem 4, *heartBeat*, se repete pela quantidade de subciclos existentes e o fim do ciclo é marcado pela mensagem 5, *waiting*, aguardando a confirmação da continuação do processamento.

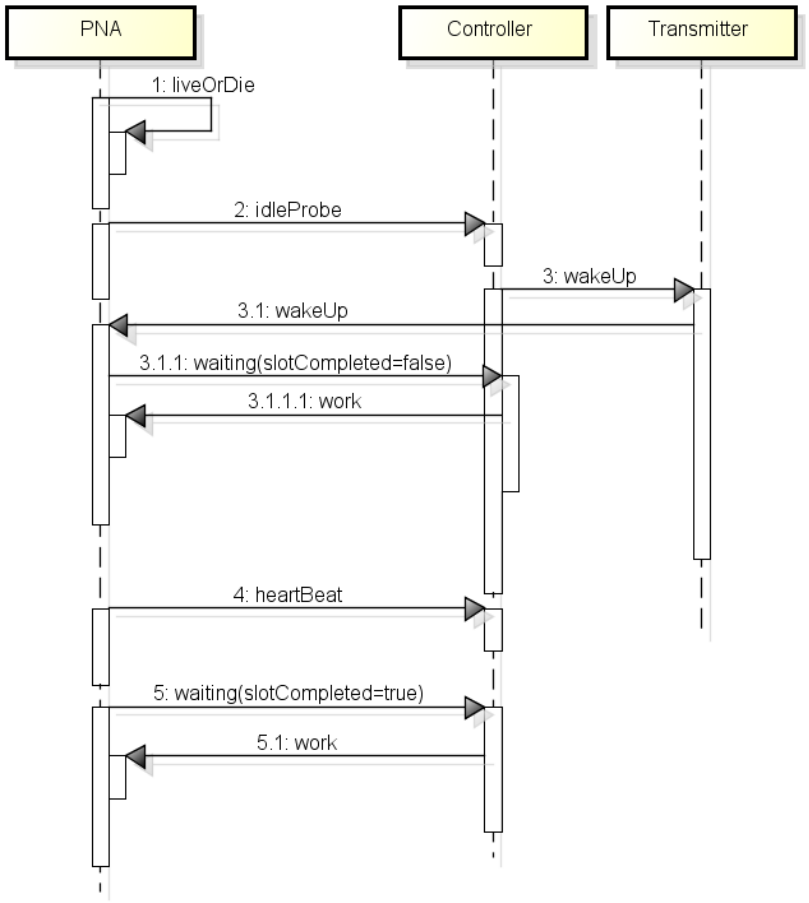


Figura 17 – Diagrama de sequência de um ciclo de processamento

Os PNA possuem *timers* para agendar as seguintes ações: *liveOrDie*, *checkAvailability*, *slot*, *heartbeat*. O *liveOrDie timer* é agendado no instante em que o PNA inicia, para cada PNA, e após a execução da rotina, o mesmo *timer* é agendado para um momento futuro dentro do ciclo, de forma aleatória, ver evento #1 e #32 na Figura 18.

O *checkAvailability timer* é um marcador de intervalos de disponibilidade, por isso, esse *timer* é agendado a cada unidade de tempo que se deseja avaliar a disponibili-

dade. Se o dispositivo estiver *on* é agendado um *checkAvailability timer* (eventos #1, #31 e #47). Caso o dispositivo entre no estado *off*, esse contador é zerado. É importante ressaltar que o gráfico da Figura 18 não é linear e que apesar de arcos de tamanhos diferentes, os intervalos entre uma verificação de disponibilidade e outra são os mesmos.

O *slot timer* possui quatro *flags* diferentes: *initial waiting, slot waiting, cancel waiting* e *idleProbe.* Na Figura 18, a flag utilizada pelo *slot timer* está explícita pela mensagem seguinte resultante. Por exemplo, o *slot Timer* do evento #11 para o evento #12 apresenta a flag *initial waiting,* pois a mensagem do evento #12 para o #15 é um *initial waiting*, em resposta a uma WM. Ao mesmo tempo o PNA agenda um *slot timer* com a *flag cancel waiting* e *timeout* configurado, para voltar ao estado *idle,* caso não seja recebida uma *workMsg* ou uma *dieMsg*. Se uma das duas chega ao PNA, o *slot timer* com flag *cancel waiting* é cancelado. O *slot timer* com flag *slot waiting* é agendado no início do ciclo para o final, eventos #18 para #49, ao receber uma *workMsg* (#15 a #18). Por fim, o *slotTimer* com flag *idleProbe,* mensagem do evento #1 ao #2, como diz, resulta em uma sonda para o *Controller,* mensagem do evento #2 ao #5, em cada momento em que o PNA está disponível.

O *heartBeat timer* é agendado como resposta a uma mensagem do tipo *work*, para enviar uma mensagem do tipo *heartBeat* ao fim de um subciclo, ver eventos #14 e #15. Em seguida, a cada subciclo um novo *heartBeat timer* é agendado, eventos #19, #26, #35 e #42. O *slot timer* com flag *slot waiting* demarca o fim do último subciclo e ao mesmo tempo do ciclo inteiro.

3.5. O Papel do *Controller*

O *Controller* se comporta como um receptor de solicitações do *Provider* e se responsabiliza por iniciar e manter as instâncias nos níveis de paralelismo e vazão solicitados. Para iniciar uma instância ele inicia um *wakeup process* enviando para o *Transmitter* a aplicação contendo toda a imagem que será executada no PNA. Após a chegada da mensagem pelo canal de *broadcast,* o *Controller* se encarrega de ingressar os dispositivos nas suas instâncias, descartando os excedentes.

Para acompanhamento das tarefas que estão sendo processadas, o *Controller* salva em arquivo os registros de ciclos como um histórico e aplica suas heurísticas para o ciclo e subciclo correntes de cada instância. O PNA sabe apenas em qual ciclo está, o subciclo é controlado pelo *Controller*, visto que um PNA pode começar em qualquer

subciclo de um ciclo, pois várias WM são enviadas nesse intervalo. Esses registros contêm a contabilização de dispositivos ativos e descartados em cada subciclo para que seja feita uma avaliação da instância ao término do subciclo para configuração dos parâmetros da nova WM. Com isso, cada WM possui valores estimados nos subciclos anteriores visando a compensação no que está começando.

Internamente, assim como o PNA, o *Controller* é coordenado por *timers*. São usados os seguintes *timers* por instância: *evaluation*, *record cycle*, *change cycle*, *finish subcycle*, *send broadcast* e *schedule broadcastMessage.*

Ao iniciar uma instância, o *Controller* agenda uma avaliação inicial através do *evaluation timer* para mandar um *wakeup*, pois a avaliação é o decisor sobre a necessidade de enviar *wakeups.* Como a instância não possui nenhum dispositivo alocado inicialmente, necessariamente será enviado um *wakeup*. Em seguida o *Controller* agenda *timers* para mudar de subciclos, *timer finishSubCycle*, eventos #25, #34 e #41 e #56, na Figura 19. E simultaneamente para mudar de ciclo no fim do ciclo corrente, *timer changeCycle* como no evento #56 da mesma figura.

Para enviar uma mensagem de broadcast o *Controller* pode agendar um *timer send broadcast*, através da qual a mensagem será imediatamente transmitida de acordo com a disponibilidade do canal de *broadcast*, ou seja, se alguma mensagem estiver sendo transmitida, a mensagem agendada com este *timer* só começará a ser enviada após o término da anterior. Caso o *Controller* agende uma mensagem de broadcast para um momento posterior, ele utiliza o *timer schedule broadcastMessage.* Ao chegar o momento agendado a mensagem é tratada pelo *timer send broadcast.*

Para manter a instância atendendo à solicitação do *Client*, as avaliações são agendadas estrategicamente para um período posterior à contagem de dispositivos que ocorre em cada subciclo. Assim como ocorre o agendamento de gravação dos resultados do ciclo em arquivo para o começo do segundo ciclo posterior, possibilitando que mesmo um PNA que iniciou seu ciclo no último minuto do ciclo corrente seja capturado. Isso ocorre porque quando uma WM é enviada em um novo ciclo, os PNA que já estavam ativos e ainda não concluíram o ciclo anterior têm seus *heartBeats* contados, até que seja feito o registro dos resultados deste ciclo.

Mais detalhes das operações realizadas pelo *Controller* são explicitadas no próximo capítulo.

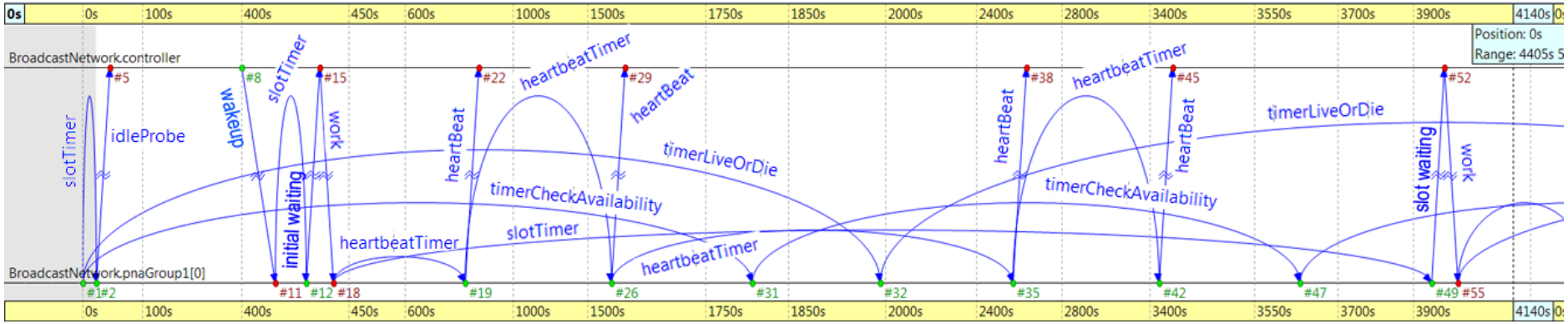


Figura 18 – Diagrama de sequência dos timers internos ao PNA em sincronia com as mensagens externas

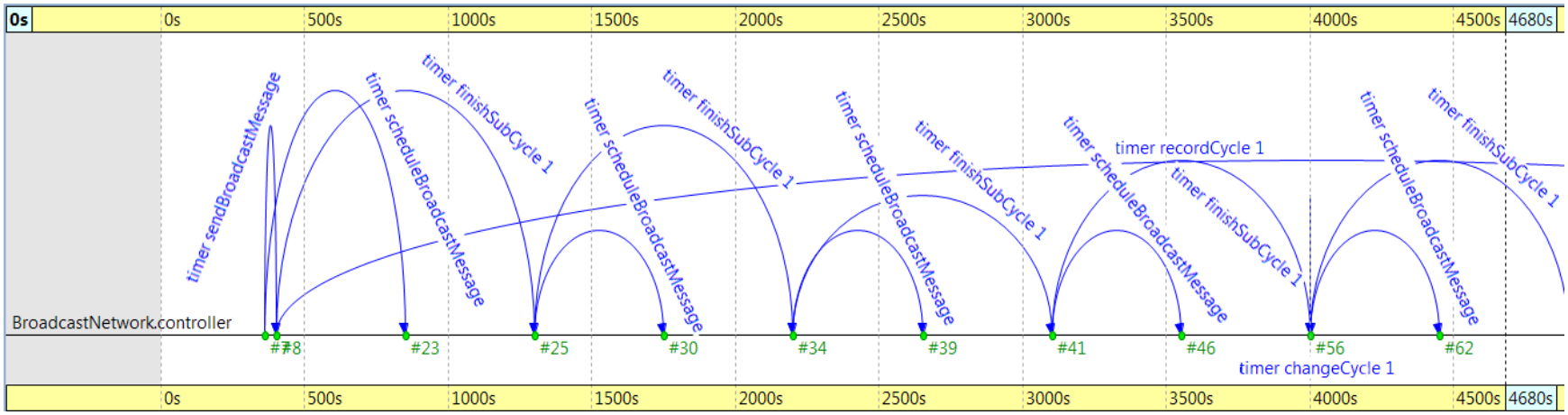


Figura 19 – Diagrama de sequência dos timers internos ao Controller

Capítulo 4. Heurísticas para Coordenação de Infraestruturas Computacionais via Broadcast

Este capítulo trata do núcleo do modelo proposto neste trabalho. Que são as heurísticas implementadas relativas ao aprovisionamento de recursos levando em consideração a volatilidade observada na rede de *broadcast* e os métodos de estimativa e seleção de PNAs.

O problema de coordenar uma infraestrutura baseada em recursos voláteis através de um canal de broadcast é definido da seguinte forma: dados um conjunto P de processadores voláteis, um *Controller* c, um canal de *broadcast* unidirecional b, um conjunto de canais de retorno bidirecionais R e um conjunto de *Backends* B, tais que processadores pertencentes a P recebam tarefas de c, como manter ativo um subconjunto dos processadores de P com tamanhos solicitados S por um conjunto K de *Clients*, tirando proveito da comunicação em massa provida pelo canal b e usando o conjunto R para outras mensagens de controle.

Esses processadores são recursos alocados para processar aplicações paralelas, são eminentemente voláteis e alocar um determinado conjunto de recursos para processar tarefas em paralelo implica que, ao longo do tempo, o conjunto sofrerá redução no seu tamanho.

Para resolver este problema é necessário reparar a perda esperada de nós através de alguma estratégia de antecipação ou compensação com os chamados *algoritmos compensatórios*.

As heurísticas de aprovisionamento são algoritmos que buscam sempre atender aos acordos de nível de serviço (SLA) estabelecidos. E atender a esses acordos significa manter o nível de paralelismo real (P_R), ou número de processadores ativos, o mais próximo possível do paralelismo solicitado (P_S).

Os desafios a serem resolvidos para que esses acordos sejam sempre respeitados são: 1) manter P_R próximo a P_S; 2) estimar o tamanho de dispositivos disponíveis e compensar a taxa de falha dos dispositivos. Para isso, são apresentadas duas heurísticas de aprovisionamento na Seção 4.3, que combinadas com as técnicas de estimativas de PNA

apresentadas na Seção 4.1 e as técnicas de seleção de PNA da Seção 0, são nossas soluções para os desafios respectivamente citados.

4.1. Métodos de Estimativa de PNAs

O primeiro desafio a ser resolvido na coordenação de sistemas distribuídos que utilizam recursos voluntários, assim como os sistemas OddCI, é a necessidade de ter uma aproximação da quantidade de dispositivos disponíveis (D_D), ou seja, no estado *idle*, prontos para serem alocados em uma nova instância. Com a posse dessa informação é possível seguir uma abordagem seletiva considerando apenas a quantidade de dispositivos solicitados para cada instância, evitando gargalos no *Controller* por excesso de PNAs solicitando a participação da instância. Isso ocorre porque o canal de broadcast é unidirecional e permite atingir e descobrir todos os recursos simultaneamente. Do contrário, seria necessário: a) conhecer todos os recursos previamente e b) convidar cada um individualmente. Para prover essa informação, apresentamos duas abordagens: uma informativa e outra exploratória.

Em ambas abordagens as WM possuem um fator probabilístico de adesão usado pelo PNA para decidir se deve responder ou não a essa mensagem. Esse fator é utilizado na função discreta de distribuição de probabilidade *bernoulli*(p), onde $0 \leq p \leq 1$, p é o fator de probabilidade do resultado ser verdadeiro (1) e $1 - p$ é a probabilidade do resultado ser falso (0). O PNA responde apenas quando o resultado for verdadeiro. Com base nisso, aproximadamente $p \times D_D$ respondem à WM, com maior precisão quanto maior for D_D.

4.1.1. Abordagem Exploratória

Na abordagem exploratória, inicialmente, não há nenhum tipo de aproximação prévia para D_D. Por isso, a primeira WM é enviada para que os PNAs respondam a uma instância auxiliar de coordenação, que não é considerada como uma instância real, ou seja, não há uma requisição de processamento para essa instância, e serve somente para estimar D_D utilizando um fator probabilístico baseado na média de audiência para o horário em questão, para se tentar evitar uma sobrecarga de respostas no *Controller*. E com base nos dispositivos que respondem (D_R), pode-se fazer a seguinte aproximação:

$$D_D = \frac{D_R}{p}$$

Com um valor mais próximo da realidade, pode-se determinar um fator p mais adequado com a quantidade de dispositivos que se deseja alocar para cada instância, sem ser preciso que todos os D_D respondam à WM, exceto quando o fator probabilístico for igual a 1. O cálculo do fator probabilístico do próximo subciclo é realizado com base no subciclo anterior, conforme o Algoritmo 2:

Algoritmo 2 – ExploratoryCalcNextFactor

```
prevRespondingPNAs = Número de PNAs que responderam a WM no subciclo anterior
prevAllocatedPNAs = Número de PNAs alocados no subciclo anterior
prevDiscardedPNAs = Número de PNAs descartados no subciclo anterior
prevFactor = O fator probabilístico de adesão gerado por esse algoritmo para o subciclo anterior
factorIncrement = Um fator de incremento gradual baseado na dimensão da instância
requestingDevices = Quantidade de dispositivos a serem requisitados no subciclo corrente

1       prevRespondingPNAs = prevAllocatedPNAs+ prevDiscardedPNAs
2       IF prevRespondingPNAs = 0 and prevFactor < 1 THEN
3               nextFactor = prevFactor + factorIncrement
4       ELSE
5               IF prevFactor = 0 THEN
6                       estimatedPNAs = prevRespondingPNAs
7               ELSE
8                       estimatedPNAs = respondingPNAs / prevFactor
9               ENDIF
10              nextFactor = requestingDevices/estimatedPNAs
11              IF nextFactor > 1 THEN
12                      nextFactor = 1
13              ELSE
14                      IF nextFactor < 0 THEN
15                              nextFactor = 0
16                      ENDIF
17              ENDIF
18      ENDIF
19      RETURN nextFactor
```

O Algoritmo 2 utiliza as informações do subciclo anterior, linha 1, obtendo a quantidade de PNAs que responderam à WM para calcular o fator probabilístico de adesão para o subciclo que está começando. São tomadas as devidas precauções para evitar a divisão por zero, linha 5. A linha 10 é uma razão entre o número de dispositivos a serem requisita-

dos no subciclo corrente e a quantidade estimada de PNA com resultados do subciclo anterior, linha 8.

O algoritmo para estimativa da quantidade de dispositivos a serem requisitados no subciclo corrente, *requestingDevices*, é apresentado na seção 4.3, Algoritmo 3.

4.1.2. Abordagem Informativa

Na abordagem informativa todos os dispositivos disponíveis enviam frequentemente sondas de disponibilidade. O *Controller* recebe essas sondas e as contabiliza, podendo assim calcular um fator probabilístico de adesão apropriado para que apenas o número requisitado de dispositivos de cada instância responda a WM. Desta forma, ele impede que cada WM das diversas instâncias desperte um número de dispositivos muito acima do que o *Controller* possa atender, ou muito abaixo de P_S. O cálculo do fator para o estimador informativo é simplesmente dividir a quantidade de dispositivos solicitados no subciclo corrente pela quantidade de dispositivos que enviaram uma sonda de disponibilidade (*idleProbe*).

4.2. Métodos de Seleção de PNAs

A utilização de métodos de predição para suportar mecanismos que assegurem a disponibilidade coletiva (*collective availability*) de uma coleção volátil de recursos foi estudada por Andrzejak, *et al*., (2008). O estudo mostra que através de previsão, métodos acurados podem ser usados para garantir que um subconjunto δ qualquer de nós em um conjunto volátil Π esteja disponível durante um período ΔT ao custo de uma sobrecarga de controle razoável. As técnicas de predição estudadas foram bem sucedidas também para avaliar o custo de disponibilidade do subconjunto em termos de nível de redundância necessário e da sobrecarga de migração entre recursos não dedicados e dedicados para compensar a perda de nós.

A taxa de sucesso (*success rate*) obtida quando se tenta manter um subconjunto δ de dispositivos disponíveis em um dado período é dependente do tempo médio de disponibilidade dos dispositivos do conjunto volátil (*historical turnover rate*) e do tamanho de δ, mas pode ser equilibrada através de um nível adequado de redundância R através da alocação de $|\delta| + R$ recursos computacionais (vide mais detalhes na Seção 4.3). Os resultados de An-

drzejak, *et al*., (2008) indicam que a solução mais prática para controle da disponibilidade coletiva é uma combinação de uma abordagem de previsão simplificada com o ranqueamento dos dispositivos de acordo com a média da sua disponibilidade histórica.

4.2.1. Seleção de PNAs por Ranqueamento

Com base no que foi dito, uma sequência de bits 0 ou 1 pode 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 permitir ou não a sua escolha para o atendimento de requisitos de disponibilidade específicos. Em nossa abordagem, uma variação mais escalável desse método é obtida através do cálculo e manutenção das informações históricas e do ranqueamento da disponibilidade média nos próprios dispositivos (PNA), dessa forma, o *Controller* não precisa armazenar informações sobre a disponibilidade de cada PNA.

Para isso, na abordagem de seleção por ranqueamento é utilizado um alvo de ranqueamento que é a quantidade de intervalos que se espera que o PNA permaneça ativo em média, o qual é informado pelo *Controller* juntamente com os outros requisitos para o dispositivo na WM e, considerando o seu histórico de disponibilidade, o PNA decide se juntar-se-á ou não à instância.

Como este método de seleção faz com que menos dispositivos respondam, além dos que já tiveram um fator probabilístico de adesão positivo resultante do método de estimativa de PNAs, não pode ser utilizado em conjunto com o método de estimativa exploratório, já que este estimador usa apenas a quantidade de dispositivos que responderam às WMs para tal estimativa. Já o estimador informativo pode ser utilizado com o ranqueamento, pois utiliza as sondas de disponibilidade que são independentes das respostas às WMs.

4.2.2. Seleção de PNAs por Ordem de Descoberta

Para efeitos de comparação, também usamos uma abordagem de seleção por ordem de descoberta ou ordem de chegada. Os primeiros P_S dispostivos que respondem à WM são alocados na instância. Ou seja, cada dispositivo que envia uma *waitingMsg* passa a fazer parte

imediatamente da instância até atingir a quantidade P_S recebendo uma *workMsg*, depois disso, todos são descartados recebendo uma *dieMsg*.

4.3. Heurísticas de Aprovisionamento de PNAs

A alocação inicial de recursos para criar uma instância com $|\delta| +$ R dispositivos é realizado em um único passo pelo *Controller* que envia para os dispositivos as instruções necessárias, incluindo o ranqueamento desejado, através de uma WM. Este processo será 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 de R é definido dinamicamente em cada avaliação da instância considerando a taxa de perda de nós observada (volatilidade).

Entretanto, uma WM pode ativar um número muito maior ou muito menor de dispositivos do que o necessário, a depender da disponibilidade instantânea dos dispositivos, que podem já estar ociosos, desligados, servindo a outra (ou à mesma) instância ou podem ter sido ligados após a última WM. Qualquer excedente de PNAs precisa ser descartado pelo *Controller*, da mesma forma que uma quantidade menor demandará novas tentativas de alocação.

A **heurística de paralelismo limitado** consiste em atender à solicitação de criação da instância do *Client* usando o número de dispositivos solicitados (D_S), equivalentes a P_S, como limite superior, mantendo o número de dispositivos ativos na instância (D_A), equivalente a P_R, sempre menor ou igual a D_S. Esta heurística é mais adequada quando o *Backend* possui algum tipo de limitação que só lhe possibilite atender até um número equivalente a D_S recursos simultaneamente. E para isso, as WM possuem um fator probabilístico que representa a quantidade de dispositivos a serem solicitados calculada pela diferença $D_S - D_A$. Por isso, o *Controller* nunca irá solicitar mais dispositivos que D_S. Portanto, sempre que $D_A < D_S$ haverá uma nova WM para compensar as perdas.

A **heurística de tempo mínimo de finalização** usa D_S como limite inferior para D_A. Para isso, é utilizado um cálculo que acrescenta redundâncias para o número de dispositivos a ser requisitado de acordo com a quantidade de dispositivos que estão faltando e com o fator de volatilidade observado em toda a rede de *broadcast*, que representa a quantidade de

dispositivos que estão falhando em toda a rede considerando o intervalo de um ciclo. Este fator de volatilidade é atualizado por todas as instâncias ativas a cada avaliação de subciclo, conforme o Algoritmo 3, linha 12:

Algoritmo 3 – minTimeNextRequestingDevices

```
cycle = Ciclo corrente que está sendo avaliado
prevPrevCycle = Ciclo anterior ao anterior do ciclo corrente
currentSubCycle = subciclo corrente do ciclo
prevSubCycle = subciclo anterior ao subciclo corrente

1       IF (cycle.id = 0 OR cycle.id = 1) OR currentSubCycle.id <> 0 THEN
2               IF prevSubCycle.deficitDevices < 0 THEN
3                       RETURN prevSubCycle.failedDevices
4               ELSE
5                       IF (cycle.id = 0 OR cycle.id = 1) and cycle.currentSubCycle.id = 0 THEN
6                               RETURN prevSubCycle.deficitDevices/(1-volatilityFactor)
7                       ELSE
8                               RETURN prevSubCycle.deficitDevices+prevSubCycle.failedDevices*3
9                       ENDIF
10              ENDIF
11      ELSE
12              volatilityFactor = 1 - prevPrevCycle.completedSlots/prevPrevCycle.startedSlots
13              RETURN (prevPrevCycle.startedSlots*instance.requestedDevices/
                prevPrevCycle.completedSlots) - prevSubCycle.workingPNAs
14      ENDIF
```

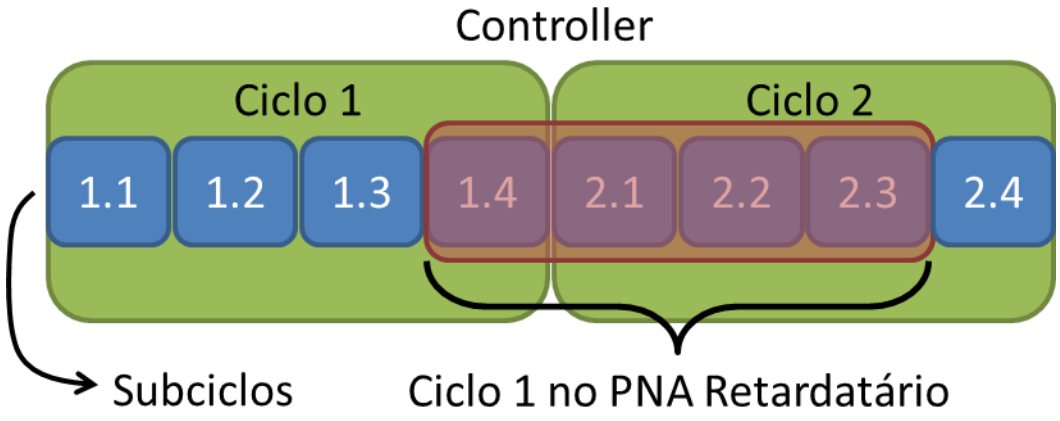


Figura 20 – Visão do Ciclo em PNAs retardatários

O Algoritmo 3 se baseia no fato de que um ciclo não termina até que comece o ciclo posterior ao posterior. Por exemplo, caso um ciclo seja dividido em 4 subciclos (a quantidade de subciclos é variável), conforme a Figura 20, o *Controller* pode alocar dispositivos para o ciclo 1 até o seu quarto subciclo (1.4). Isso fará com que o dispositivo esteja associ-

ado ao ciclo 1 quando o ciclo 2 já foi iniciado no *Controller* e o dispositivo só terminará o ciclo 1 quando o terceiro subciclo do ciclo 2 for concluído no *Controller* (2.3). Essa informação é importante, pois os PNAs retardatários (dispositivos que estão trabalhando no ciclo anterior ao ciclo corrente no *Controller*) devem ser considerados na contagem de dispositivos que conseguiram finalizar slots de processamento, por isso, o PNA não faz o registro do seu subciclo correspondente, apenas do seu ciclo.

Portanto, o algoritmo começa verificando se o ciclo é o 0 ou o 1, pois ainda não há, até que estes terminem, informação correta do número de slots completados, ou seja, não se sabe qual é a volatilidade da rede se esta for a primeira instância que está sendo alocada. Se não for a primeira instância, e for um dos dois primeiros ciclos e primeiro subciclo, a variável global *volatilityFactor* é aproveitada na linha 6 do algoritmo tendo sido preenchida por outra instância ou por ela mesma na linha 12. Esta variável começa com 0 e só é alterada na linha 12, quando o 3º ciclo está começando (cycle.id = 2).

Os *deficitDevices* são a quantidade de dispositivos que faltam para atingir o número requisitado de dispositivos para a instância, baseado no número de dispositivos ativos contados pelas sondas *heartBeatMsg* do subciclo anterior. Ao iniciar, como não houve contagem de dispositivos ativos, a variável *deficitDevices* é equivalente à quantidade de dispositivos solicitados. Um valor negativo para esta variável significa que o número de dispositivos ativos está acima do solicitado, que é o que se espera da heurística de tempo mínimo de finalização (estar sempre acima do solicitado), sendo necessário apenas solicitar o que falhou de um subciclo para o outro, linhas 2 e 3, para manter a instância com a vazão desejada.

Os *failedDevices* representam a quantidade de dispositivos que falharam entre o subciclo anterior e o subciclo corrente. É calculado por:

$$\text{failedDevices} = \text{prevWorkingPNAs} + \text{currentAllocatedPNAs} - \text{currentWorkingPNAs}$$

onde *WorkingPNAs* são os dispositivos ativos no subciclo anterior ou no corrente e *currentAllocatedPNAs* são os dispositivos que foram alocados apenas no subciclo corrente.

Na linha 6, o *volatilityFactor* funciona como uma redundância para solicitação de dispositivos acima do número requisitado levando em conta o que se espera que falhe no próximo ciclo, para os dois primeiros ciclos, pois a partir do terceiro, a informação de volatilidade da rede já está disponível pelo primeiro ciclo. Na linha 8, essa redundância é base-

ada nos dispositivos que falharam no subciclo anterior. Na linha 13, o cálculo dos dispositivos a serem requisitados é feito por uma regra de três:

$$\frac{cycle.requestingDevices}{instance.requestedDevices} = \frac{prevPrevCycle.startedSlots}{prevPrevCycle.completedSlots}$$

onde:

$prevPrevCycle.startedSlots$ = Slots iniciados do ciclo anterior ao anterior do ciclo corrente

$prevPrevCycle.completedSlots$ = Slots completados do ciclo anterior ao anterior do ciclo corrente

cycle.requestingDevices = a quantidade de dispositivos a ser solicitada no ciclo corrente

$instance.requestedDevices$ = quantidade de dispositivos que devem ser completados no ciclo corrente = Quantidade requisitada pela instância.

Isolando a variável que queremos descobrir, cycle.requestingDevices, usando a razão de volatilidade do ciclo anterior e subtraindo a quantidade de dispositivos que ainda estão ativos (*prevSubCycle.workingPNAs*), temos a linha 13 do algoritmo.

Capítulo 5. Avaliação de Desempenho

Para responder à questão de pesquisa levantada na Seção 1.3 "*Como alocar e manter uma infraestrutura confiável de processamento paralelo coordenando recursos distribuídos, não dedicados e de alta volatilidade utilizando um canal de broadcast e canais de retorno como a Internet?*", foram criadas as heurísticas do Capítulo 4. Este capítulo descreve como essas heurísticas foram avaliadas através de experimentos baseados em simulação.

5.1. Descrição dos experimentos

Os recursos de processamento (PNA) constituintes de sistemas OddCI são extremamente voláteis e distribuídos, o que significa que uma instância alocada em algum momento terá o seu tamanho alterado. As heurísticas apresentadas são soluções que buscam compensar esse problema natural de uma rede de nós voluntários em cenários diferentes.

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:

- *Backend de capacidade limitada* – cenário tratado pela **heurística de paralelismo limitado**, onde o número de dispositivos solicitados é o limite máximo para a quantidade de dispositivos ativos na instância;
- *Aplicações sensíveis ao tempo* – cenário tratado pela **heurística de tempo mínimo de finalização**, onde o número de dispositivos solicitados é o limite mínimo para a quantidade de slots completados na instância por ciclo.

Os principais gargalos a serem testados foram: 1) a disponibilidade de dispositivos para atendimento das solicitações e 2) a concorrência pelo uso do canal de transmissão em *broadcast.* O primeiro gargalo é dependente do universo de dispositivos, *U*, das solicitações, da disponibilidade inicial e da volatilidade (esses dois últimos são parâmetros de entrada no algoritmo *liveOrDie* da Seção 3.4. Enquanto que o segundo gargalo depende principalmente da quantidade de instâncias simultâneas (*S*) e o tamanho da imagem (*I*).

Para medir os níveis de estresse sobre a disponibilidade foi fixado um pico de demanda (*P*), que representa o máximo da soma de dispositivos alocados para todas as instâncias em um dado momento de um período de observação. A partir de *P*, os *workloads* de cada experimento foram construídos de forma relativa usando *S* e a duração das instâncias em quantidade de *slots* (*D*). Assim, o *workload* de cada experimento é baseado na sua configuração e formado por *S* instâncias simultâneas iguais, todas iniciando no mesmo momento *t*, que representa o tempo mínimo para realizar uma primeira contagem do total de dispositivos disponíveis com o método de estimativa informativo. Cada instância solicita a mesma quantidade de dispositivos ($q_j = \frac{P}{S}$) pelo mesmo intervalo de tempo ($l_j = D \times \sigma$), onde σ é a duração de um *slot*. O universo de dispositivos (*U*) é obtido pela aplicação de um fator de contenção, *c*, sobre *P*: $|U| = c \times P$ (um fator 2 equivale a uma população com o dobro da quantidade operacional mínima, um fator 3, ao triplo, e assim por diante), para medir o quanto é necessário de redundância relativa ao que foi solicitado e considerando a volatilidade.

Foi feita uma verificação do projeto de simulação através da comparação dos resultados simulados de casos extremos com os resultados esperados, que 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 10 a 100 instâncias com um total de 1.000 e 1.000.000 de *slots* de processamento.

Além dos testes acima, a consistência de saídas do simulador foi verificada para cada cenário, tanto com relação à adequação das respostas para cada configuração dos parâmetros de configuração, quanto com relação ao estado interno das variáveis do simulador em cada momento do período de observação. Uma trilha de auditoria (*log)* com registros exclusivos foi criada apenas para subsidiar esta fase de verificação.

Tabela 3 – Testes para aferição do simulador OddCI-Sim

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 criada.
2	*P*	0%	0%	O resultado obtido foi idêntico ao do teste #1.
3	10*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	*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*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 nenhum dispositivo ativo, este quadro se manteve durante o período de observação levando a um resultado similar ao do teste #1.
8	*P*	100%	0%	O resultado obtido foi idêntico ao do teste #7.
9	10*P*	100%	0%	O resultado obtido foi idêntico ao do teste #7.
10	0	100%	100%	O resultado obtido foi idêntico ao do teste #1.

A volatilidade de 100% significa que aproximadamente todos os dispositivos ativos mudarão de estado, sendo desligados, e a mesma quantidade será reposta por outros dispositivos, dificultando a finalização dos *slots* de processamento e aumentando a duração das instâncias.

Também foram feitos testes para verificação da quantidade máxima de recursos, sendo possível simular mais de 2 milhões de dispositivos, onde o fator limitante é a memória RAM disponível do computador que executa a simulação, nesse caso, os testes foram feitos com uma máquina de 12GB de memória RAM.

Para a avaliação de desempenho e os parâmetros apresentados na seção a seguir usamos a abordagem de estimação de PNAs informativa e o método de seleção por descoberta.

5.2. 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* (*V*), *Tamanho da População* ($|U|$), *Tamanho da Imagem* (*I*), *Instâncias Simultâneas* (*S*) e *Duração da Instância* (*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 considerados dois valores diferentes: *pequeno* (representativo do tamanho de módulos clientes de aplicações com 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 KB de tamanho, enquanto que as imagens do tipo grande possuem tamanho de 5MB. Os níveis atribuídos para os demais fatores em cada DoE estão apresentados na Tabela 4 e na Tabela 5.

Tabela 4 – DoE 2^k: Fatores, níveis e efeitos para o cenário Tempo Mínimo de Finalização

Fator	Baixo	Alto	Efeito Estimado	Soma dos Quadrados	% Contribuição
A: Volatilidade (*V*)	5%	75%	-0,3335	0,8896	28,41
B: População ($\|U\|$)	$(1 + V) \times P$	$10P$	0,2672	0,5710	18,24
C: Tamanho da imagem (*I*)	512 KB	5 MB	-0,1667	0,2223	7,10
D: Instâncias Simultâneas (*S*)	10	100	-0,1729	0,2392	7,64
E: Duração da Instância (*D*)	10	100	-0,0184	0,0027	0,09

Tabela 5 – DoE 2^k: Fatores, níveis e efeitos para o cenário Paralelismo Limitado

Fator	Baixo	Alto	Efeito Estimado	Soma dos Quadrados	% Contribuição
A: Volatilidade (*V*)	5%	75%	-0,2216	0,3927	16,17
B: População ($\|U\|$)	$(1 + V) \times P$	$10 \times P$	0,0449	0,0161	0,66
C: Tamanho da imagem (*I*)	512 KB	5 MB	-0,2327	0,4331	17,83
D: Instâncias Simultâneas (*S*)	10	100	-0,2412	0,4653	19,16
E: Duração da Instância (*D*)	10	100	0,0080	0,0005	0,02

A variável de resposta considerada para o cenário "com aplicações sensíveis ao tempo" foi a *vazão média* ($\overline{\Phi}$) das instâncias, que representa a quantidade média de *slots* completados por ciclo para as instâncias simultâneas. Essa métrica é dada por:

$$\overline{\Phi} = \frac{\sum_{j=1}^{S} \frac{\mathbb{C}_j}{Dq_j}}{S},$$

onde $\mathbb{C}_j$ é uma função que retorna a quantidade de *slots* completados para a instância *j*. O valor de referência para $\overline{\Phi}$ é 1, ou seja, o objetivo é que essa métrica esteja o mais próximo possível de 1. E q_j é a quantidade de dispositivos requisitados por ciclo para a instância *j*.

Para o cenário de *paralelismo limitado* foi escolhida a variável de resposta *paralelismo médio* ($\overline{\Pi}$) das instâncias, o qual representa a relação entre a quantidade média efetiva de dispositivos fornecida por ciclo e a quantidade de instâncias simultâneas. Esta métrica é dada por:

$$\overline{\Pi} = \frac{\sum_{j=1}^{S} \frac{\mathbb{A}_j}{q_j}}{S},$$

onde $\mathbb{A}_j$ é uma função que retorna a quantidade média de dispositivos ativos na instância *j*. O valor de referência de $\overline{\Pi}$ também é 1.

Foram conduzidas várias repetições dos 32 experimentos previstos no DoE realizado para cada um dos cenários considerados para obter médias com 95% de confiança com erro máximo de 5% para mais ou para menos. A contribuição de cada fator em cada cenário é mostrada na Tabela 4 (cenário de *Aplicações Sensíveis ao Tempo*) e Tabela 5 (cenário *de Backend de capacidade limitada*).

No cenário de *tempo mínimo de finalização*, os fatores da *Volatilidade* e do *Tamanho da População* foram preponderantes com participação de 28,41% e 18,24%, respectivamente (Tabela 4). Enquanto que no cenário de *paralelismo limitado*, além da *Volatilidade*, que responde por 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).

Tabela 6 – Parâmetros usados nas simulações

Parâmetro	**Cenário *Tempo Mínimo de Finalização***	**Cenário *Paralelismo Limitado***
Pico de Demanda (*P*)	10.000 dispositivos	10.000 dispositivos
Taxa Canal Direto	1 Mbps	1 Mbps
Taxa Canal de *Broadcast*	1 Mbps	1 Mbps
Duração do *slot* de processamento (σ)	1 hora	1 hora
Retardo Máximo	5 segundos	5 segundos
Disponibilidade Inicial	100% da população	100% da população
Duração da Instância (*D*)	10 *slots*	10 *slots*
Instâncias Simultâneas (*S*)	50 instâncias	{20, 40, 60, 80} instâncias
Tamanho da imagem (*I*)	2,5 MB	{1 MB, 2 MB, 3 MB, 4 MB }
População (|*U*|)	{2.*P*, 3.*P*, 4.*P*, 5.*P*, 6.*P*, 7.*P*, 8.*P*, 9.*P*}	10.*P*
Volatilidade (*V*)	{20%,30%,40%,50%, 60%,70%,80%,90%}	{20%,30%,40%,50%, 60%,70%,80%,90%}

Como resultado da análise dos efeitos através de ANOVA (Jain, 1991), o F-Value de 164,4793 (*Tempo Mínimo de Finalização*) e 252,9781 (*Paralelismo Limitado*) 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.

Para a realização das simulações, os valores dos parâmetros que não afetaram o comportamento da variável de resposta foram ajustados para os valores médios entres os níveis "Alto" e "Baixo" usados em cada DoE[3]. Para os fatores mais relevantes: *Volatilidade* e *Tamanho da População* (*Tempo mínimo de Finalização*) e *Volatilidade*, *Tamanho da Imagem* e *Instâncias Simultâneas* (*Paralelismo Limitado*), foi aplicada uma varredura de parâmetros. Para a varredura de parâmetros não foi necessário ampliar os níveis usados no DoE, posto que já ocorreram restrições relevantes nos respectivos intervalos.

A Tabela 6 mostra como o sistema foi configurado para os experimentos dos dois cenários, usando o resultado do DoE, os valores obtidos no *testbed* real (ver seção 1.1) e

[3] 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.

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.

5.3. Análise dos Resultados

5.3.1. Cenário Backend de Capacidade Limitada

No primeiro experimento, realizado para o cenário *Backend de Capacidade Limitada*, o objetivo foi observar como a variação da volatilidade (*V*), da quantidade de instâncias simultâneas (*S*) e do tamanho da imagem da aplicação (*I*) impactam na manutenção da quantidade desejada de dispositivos ativos para cada instância. Para diminuir a quantidade de variáveis, fixamos a variável de contenção de dispositivos configurando a população para 10 vezes o total da demanda concomitante esperada ($|U| = 10 \times P$). Para cobrir todas as combinações dos parâmetros de entrada, foram realizados 128 replicações do experimento, repetidos até que as médias obtidas tivessem 95% de confiança e erro máximo de 5% para mais ou para menos. A métrica de interesse observada foi o paralelismo médio das instâncias, $\overline{\Pi}$. Os resultados obtidos estão exibidos graficamente a seguir.

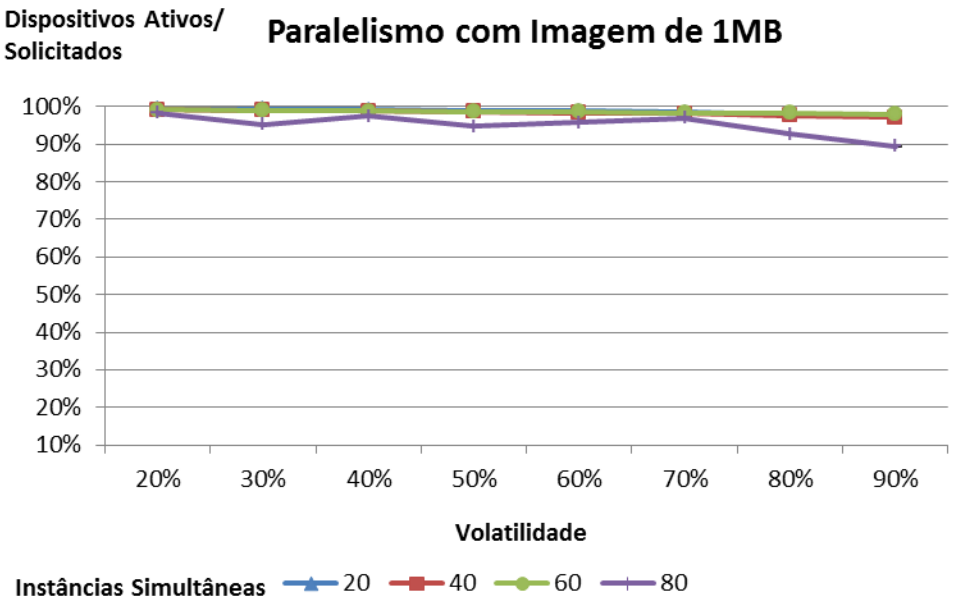


Figura 21 – Cenário Backend de Capacidade Limitada: métrica $\overline{\Pi}$ para imagem de 1MB

A Figura 21 apresenta no seu eixo vertical a relação entre a quantidade de dispositivos ativos e os dispositivos solicitados para a instância e no eixo horizontal os níveis de volatilidade inseridos no sistema. Cada curva corresponde a uma quantidade de instâncias simultâneas, 20, 40, 60 e 80. Quando lida com imagens de aplicação pequenas (1MB), o *Controller* consegue compensar a perda de dispositivos em praticamente todos os regimes de volatilidade simulados, mesmo quando coordenando muitas instâncias simultâneas. Até 60 instâncias simultâneas, o paralelismo médio é superior a 97% em todos os níveis de volatilidade, o que significa que o sistema conseguiu manter 97% do tamanho da instância solicitado. Com 80 instâncias simultâneas, o paralelismo cai para o mínimo de 89% em 90% de volatilidade.

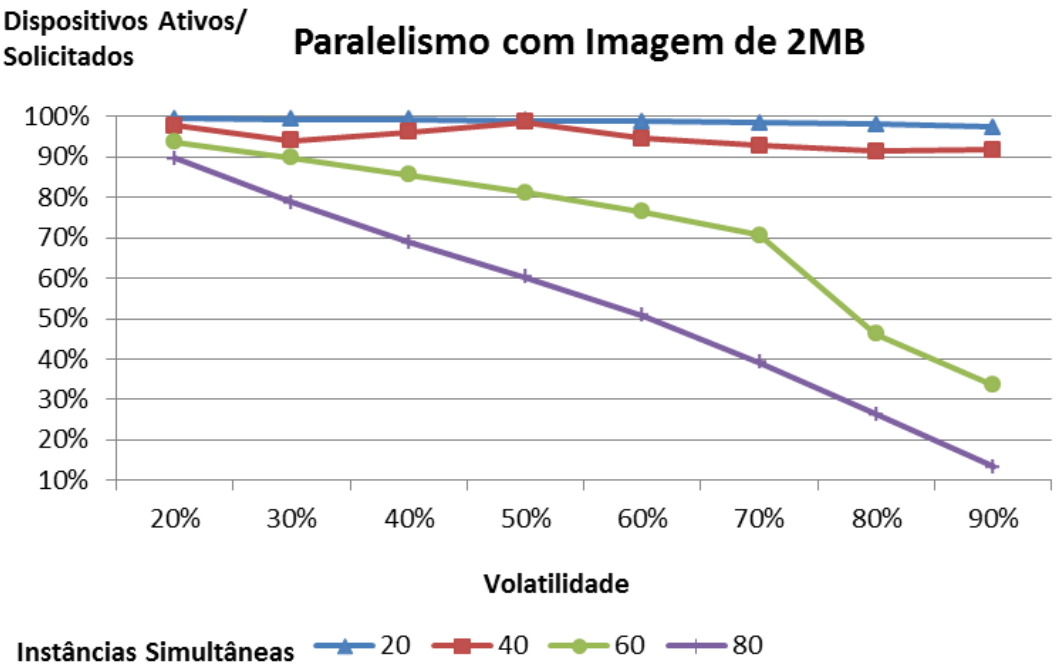


Figura 22 – Cenário Backend de Capacidade Limitada: métrica $\overline{\Pi}$ para imagem de 2MB

A Figura 22 mostra que o paralelismo médio é conservado em todas as volatilidades até as 40 instâncias simultâneas. Entretanto, com o aumento do tamanho da imagem, aumenta o tamanho da mensagem de controle correspondente (*wakeupMsg*) 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.

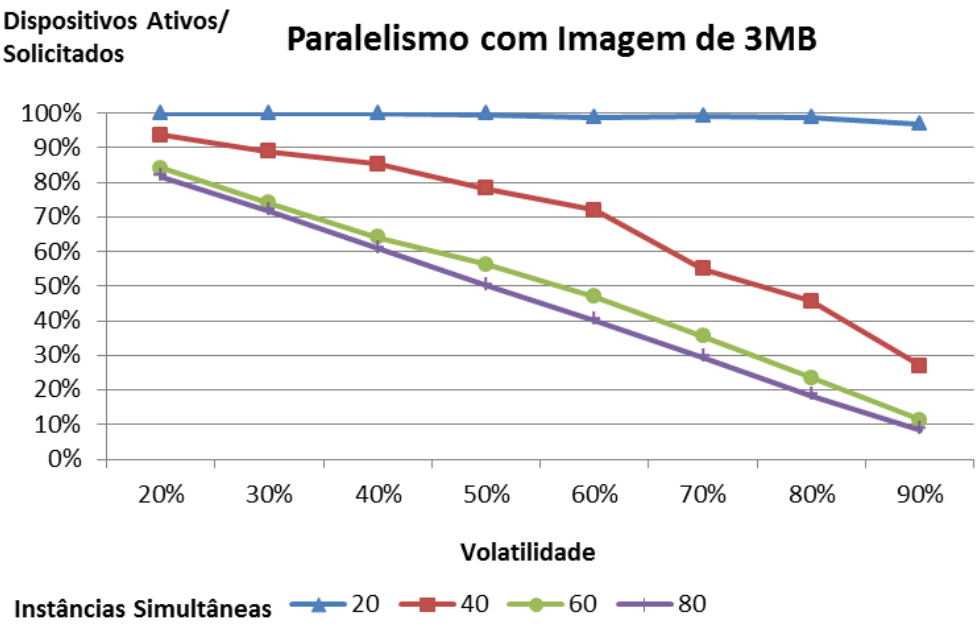


Figura 23 – Cenário Backend de Capacidade Limitada: métrica $\overline{\Pi}$ para imagem de 3MB

Este efeito 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 *Transmitter*. Esse efeito, que pode ser visualizado na Figura 21 até a Figura 24, 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.

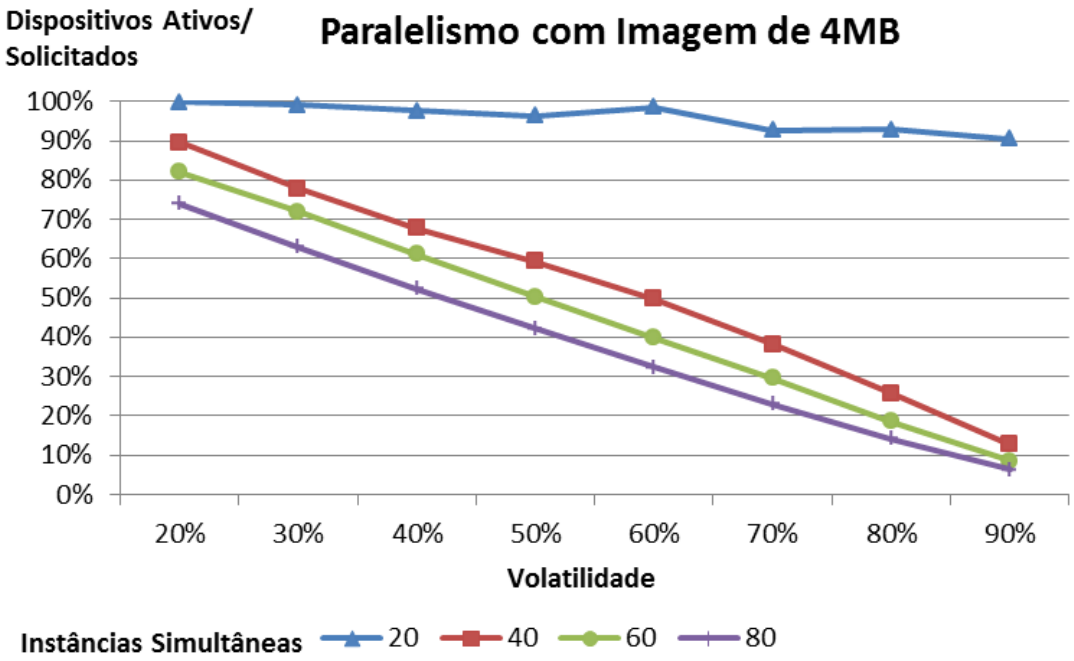


Figura 24 – Cenário Backend de Capacidade Limitada: métrica $\overline{\Pi}$ para imagem de 4MB

A Figura 24 mostra que para imagens de 4MB, o paralelismo médio só é conservado até 20 instâncias simultâneas em todos os níveis de volatilidade. Conclui-se, portanto, que para o cenário *Backend de capacidade limitada*, com imagens de até 2MB, para atender um paralelismo mínimo de 90% a quantidade máxima de instâncias simultâneas é de 40. Enquanto que, com imagens de 4MB, para atender um paralelismo mínimo de 90% a quantidade máxima de instâncias simultâneas é de 20.

5.3.2. Cenário Aplicações Sensíveis ao Tempo

No segundo experimento, realizado para o cenário de *Aplicações Sensíveis ao Tempo*, o objetivo foi observar como a variação da volatilidade (V) e do tamanho da população de dispositivos ($|U|$) 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, valor utilizado para estudar qual o nível de redundância é mais adequado para a solução de cada situação regulando o tamanho da população total, esse tamanho foi iniciado em um patamar operacional mínimo, correspondente ao pico da demanda esperada acrescido da volatilidade inserida ($|U| = P \times (1 + V)$), e foi sendo regulado pela aplicação

de um fator de contenção. Para cobrir todas as combinações dos parâmetros de entrada foram realizados 64 experimentos – repetidos até que as médias obtidas tivessem 95% de confiança e erro máximo de 5% para mais ou para menos. A métrica de interesse principal foi a mesma usada no DoE: a vazão média das instâncias, $\overline{\phi}$. Os resultados obtidos são discutidos a seguir.

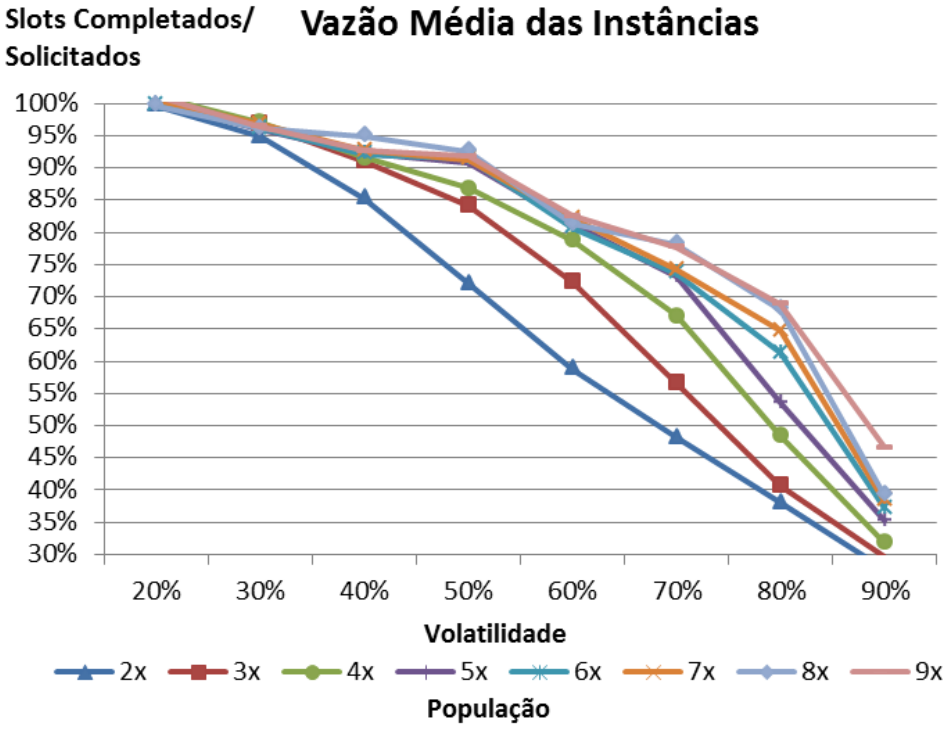


Figura 25 – Cenário Aplicações Sensíveis ao Tempo: Vazão média das Instâncias

Como ilustrado na Figura 25, a quantidade média de *slots* de processamento completados por ciclo é fortemente afetada pelo aumento da 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 15% abaixo do solicitado, com o maior fator de contenção do tamanho da população (2x). O fator de contenção 1x não foi utilizado, pois não considera redundância para o Pico de Demanda, a qual se mostra necessária em virtude da volatilidade.

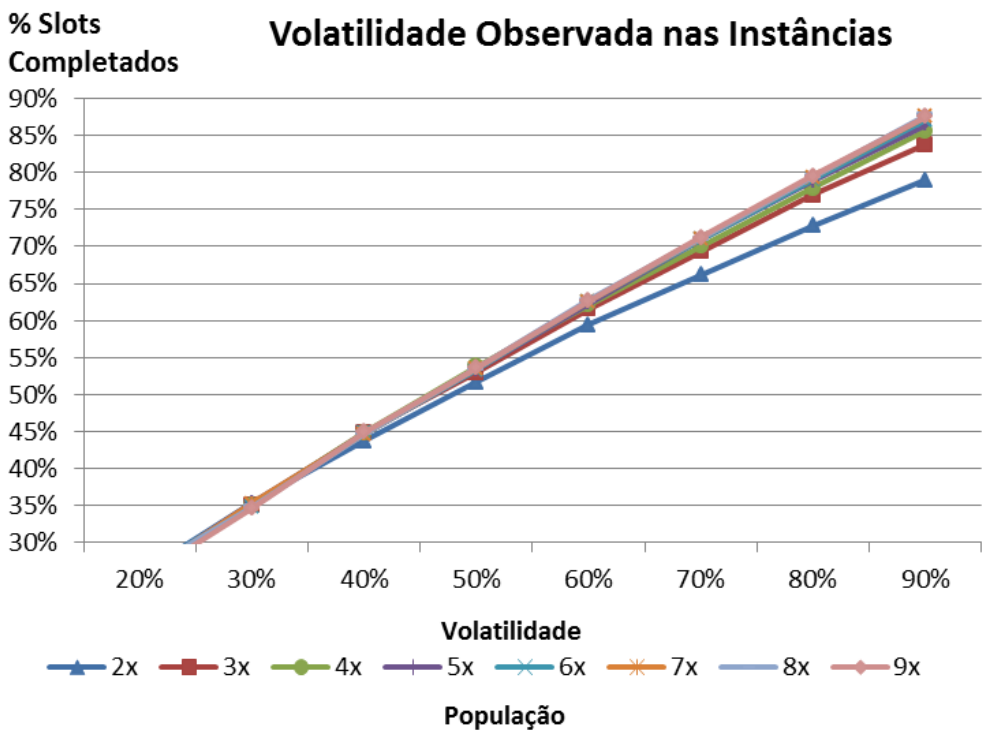


Figura 26 – Cenário Aplicações Sensíveis ao Tempo: Volatilidade Observada

Nos níveis de volatilidade adotados, o esforço de coordenação do provedor também é mantido controlado, como pode ser visto na Figura 26, que 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% (volatilidade) 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 relação entre a volatilidade inserida e a volatilidade observada é diretamente proporcional, independendo da população disponível, pois a métrica não precisa dessa variável, apenas da quantidade de dispositivos solicitados e da quantidade de dispositivos ativos em cada instante.

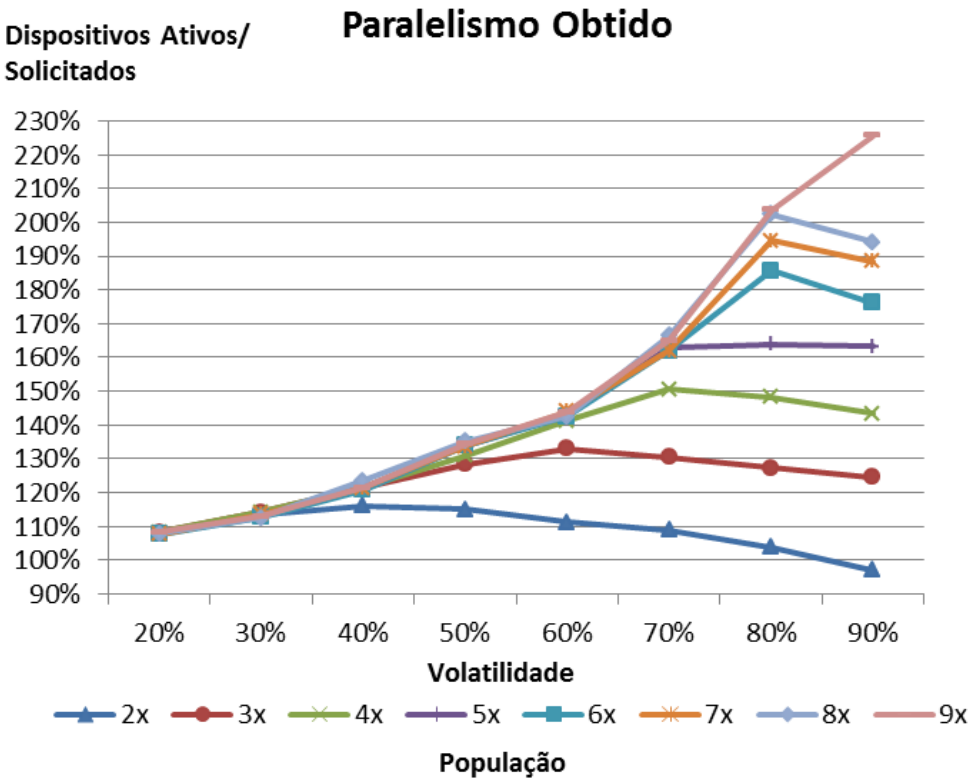


Figura 27 – Cenário Aplicações Sensíveis ao Tempo: Paralelismo

A métrica paralelismo médio das instâncias, $\overline{\Pi}$, também foi apurada para esse experimento. Pode ser visualizado na Figura 27 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. Pode-se concluir que foi necessário um paralelismo cada vez maior para o provimento da redundância necessária à medida que a volatilidade aumenta, na tentativa de respeitar a vazão mínima requerida.

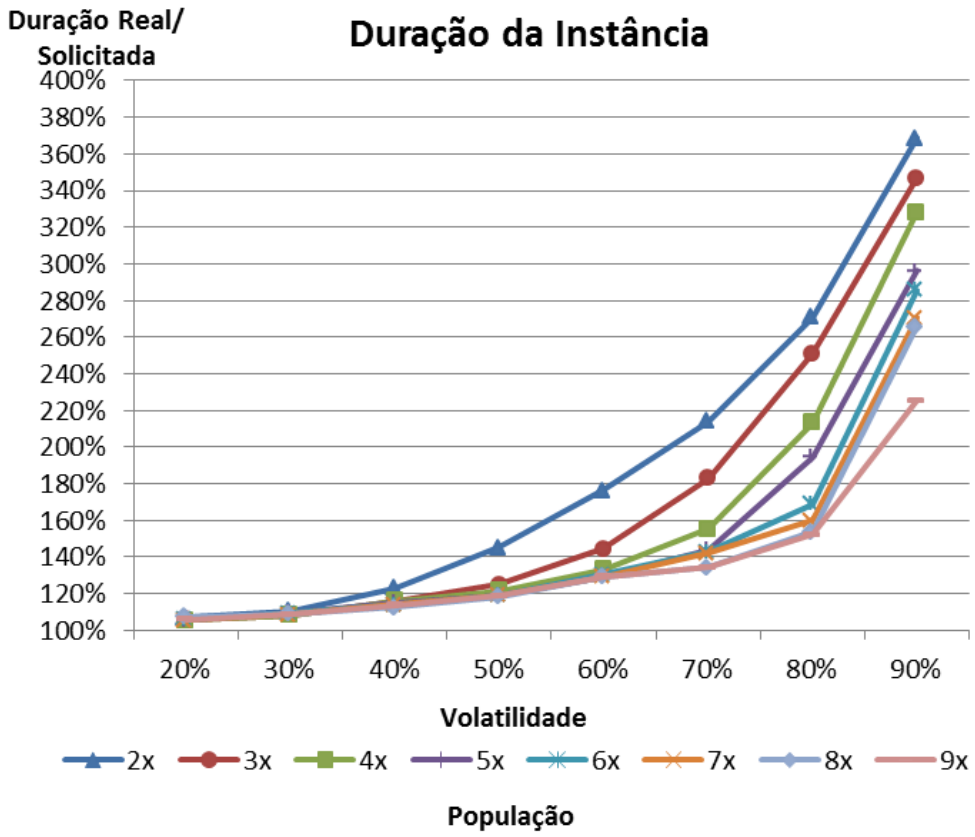


Figura 28 – Cenário Aplicações Sensíveis ao Tempo: Duração

O resultado do aumento da redundância repercute em uma atenuação dos efeitos da volatilidade sobre a vazão, com pode ser visualizado na Figura 28, na qual a duração das instâncias torna a diminuir nos cenários com menor contenção de recursos mesmo em regimes de maior volatilidade (com uma contenção 2x a população é maior que uma contenção 9x, pois em 2x há menos recursos do que em 9x). Obviamente, em contextos cuja disponibilidade de recursos não apresente restrições ao nível de redundância praticado, 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 conciliar o nível de paralelismo com a capacidade do *Backend* e com o custo operacional do provedor da infraestrutura de controle.

Capítulo 6. Trabalhos Relacionados

Neste capítulo são apresentados os trabalhos que mais se assemelham com esta proposta. Trabalhos que envolvem desde técnicas de previsão do estado dos recursos até a complexidade de infraestruturas como as Botnets.

As técnicas de previsão de estado de recursos tem resultados que servem de insumo para a fase de escalonamento de recursos da arquitetura OddCI. O uso do termo escalonamento está tradicionalmente associado à computação da ordem de execução das tarefas atribuídas a cada máquina (Braun, et al., 1998). No entanto, neste trabalho utiliza-se o termo para designar a seleção das máquinas mais disponíveis e confiáveis para a execução das tarefas, conforme a Seção 0.

Quanto à coordenação de recursos, a maioria das pesquisas foca em aumentar a vazão ou diminuir o *makespan* das tarefas diante da indisponibilidade frequente e volátil dos recursos.

Em termos de alocação de recursos sob demanda, o projeto Nephele (Warneke & Kao, 2009) em andamento é o primeiro framework de processamento de dados em paralelo a explorar explicitamente a alocação dinâmica de recursos para escalonamento e execução em ambientes de computação em nuvem.

Baseado em grafos de execução elaborados pelo usuário, o framework Nephele também possibilita, como o OddCI, alocar/desalocar automaticamente máquinas virtuais durante a execução de uma tarefa.

As estratégias de aprovisionamento propostas no OddCI para controlar o tamanho das instância e garantir que esta esteja adequada à vazão requisitada pelos clientes estão alinhadas com outras iniciativas de pesquisa.

Aron e Chana (2012) propõem um framework que oferece uma política de aprovisionamento de recurso para seu escalonamento e alocação, e demonstra que uma abordagem de aprovisionamento baseada em Qualidade de Serviço (QoS) é efetiva na minimização de custos e de tempo de rajada de submissões de aplicações.

6.1. BOINC e Botnets

No caso de aplicações Bag-of-Task (BoT), a sua eficiente execução tem sido relatada em uma variedade de infraestruturas HTC, que vão desde *grids* P2P (Litzkow, et al., 1988) (Cirne, et al., 2003) até sistemas massivos de computação voluntária como o SETI@Home (Anderson, et al., 2002) utilizando como base a plataforma BOINC (Anderson, 2004). Um exemplo deste último caso é o projeto do servidor de tarefas (*Task Server*), onde um *middleware* para computação voluntária 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).

Cirne, *et al.*, (2003) mencionam que ambientes como o BOINC apresentam indisponibilidade de recursos de até 50% e que métodos preditivos, como os citados na seção anterior, para avaliar a disponibilidade de recursos auxiliam na estruturação de redes menos voláteis.

Outra perspectiva de computação em *grid*, mas que utiliza uma abordagem ilegal para seu funcionamento são as *botnets*. *Botnets* são redes de hosts finais infectados, denominados *bots* ou zumbis, que com um malware instalado executam instruções passadas por um canal de Comando e Controle (C&C). Um *malware* é instalado no host final quando este acessa algum site ou recebe um email com um arquivo executável, por exemplo. Assim, o novo *bot* entra no canal de C&C e começa a aguardar instruções. Os *bots* são controlados por um operador humano conhecido como *botmaster* (Marupally & Paruchuri, 2010). Este canal de C&C pode ser implementado sobre protocolos de Internet como Internet Relay Chat (IRC), HTTP, DNS ou P2P.

Sánchez, *et al.*, (2009) afirmam que as *botnets* são um dos paradigmas de computação distribuída/nuvem mais bem estabelecidos e comercialmente bem sucedidos. Isso se deve, parcialmente, à facilidade de aquisição dos recursos computacionais, provando sua eficiência no controle de redes P2P com mais de 400.000 nós (McLaughlin, 2004).

6.2. Crescimento de recursos voluntários

Conforme Kacsuk (2011) existem dois tipos de *grids* computacionais, os *service grids* (SG) e os *desktop grids* (DG). Os SG geralmente são constituídos por clusters como o *grid* EGEE baseado no gLite ou por supercomputadores como o *grid* DEISA baseado no UNICORE e proveem um serviço 24/7 para uma grande quantidade de pessoas. Todavia, o uso de supercomputadores é extremamente caro, enquanto que os DG são a forma mais barata, por serem recursos já existentes, de criar e manter *grids* institucionais de larga capacidade com um número típico de recursos voluntários entre 10 mil e 1 milhão.

Toth, Mayer e Nichols (2011) desenvolveram um experimento que manipulava um roteiro de recrutamento que descrevia uma oportunidade de participar em um projeto de computação voluntária. As amostras de estudantes analisadas eram apresentadas a uma das quatro variações do roteiro, dos quais duas continham mensagens que visavam combater consequências negativas percebidas na participação de projetos de computação voluntária. Todos os roteiros usaram a descrição de computação voluntária do projeto Docking@Home como modelo. Como resultados, perceberam que a complacência para participar do projeto é minimamente afetada pelos custos de energia e desgaste dos computadores. A pouca preocupação com custos de energia se deve ao fato de que os estudantes não pagam diretamente as contas de eletricidade, o que não é a realidade da TV Digital, pois os receptores geralmente estão em residências. E o receio quanto ao desgaste dos computadores é amenizado por causa da velocidade dos *upgrades* quando estes se tornam obsoletos.

Além das questões econômicas envolvidas, o fato de somente os gerentes dos projetos sobre a plataforma BOINC submeterem tarefas é uma limitação no estímulo da participação de voluntários. Por isso, Kacsuk (2011) propõe um cenário de utilização com maior abstração para acesso aos serviços de um *grid* computacional através de portais como o P-GRADE (Németh, et al., 2004), através dos quais é possível submeter nós do fluxo de trabalho para varredura de parâmetros em uma interface web, por exemplo. Desta forma, a utilização incentiva o interesse em disponibilizar recursos para o *grid*.

Capítulo 7. Conclusões

Com a implementação da base da arquitetura apresentada por Costa, et al., (2009) como um modelo de simulação, foi possível realizar os experimentos planejados para os sistemas OddCI, o que possibilitou o melhor entendimento da dinâmica do funcionamento de toda a infraestrutura.

Foi possível determinar os fatores mais impactantes no desempenho do sistema, nos cenários distintos avaliados. No cenário *backend de capacidade limitada*, os principais fatores foram o tamanho da imagem utilizada pelas instâncias e a quantidade de instâncias simultâneas, sendo possível atender no caso mais extremo de imagens de 4MB e 80 instâncias simultâneas e volatilidades de 40% com 50% do paralelismo solicitado. Enquanto que no cenário *tempo mínimo de finalização*, os principais fatores foram a volatilidade, a população e, consequentemente, a sua disponibilidade, sendo possível atender 50 instâncias simultâneas com redução de apenas 15% da vazão média solicitada no caso de maior contenção (2x o Pico de demanda = menor população) para volatilidades de até 40%. Assim, foi possível verificar até que nível os resultados foram favoráveis em cada cenário. O que responde à pergunta levantada neste trabalho: sim, existe uma forma de coordenar uma infraestrutura de forma confiável baseada em recursos voláteis como os receptores de TV Digital.

Apesar disso, é perceptível o gargalo de transmissão no canal de *broadcast* quando o tamanho da imagem aumenta, mas o problema do limite da quantidade de instâncias simultâneas pode ser contornado, no caso do sistema brasileiro de televisão digital, com a utilização de emissoras localizadas em cidades diversas funcionando como o elemento *Transmitter* citado na Seção 3.1 sendo acionados por um ou mais *Controllers*. Desta forma, cada cidade vai representar um domínio diferente de recursos voluntários. Associadamente, a inclusão de mecanismos de controle de admissão de instâncias pode aperfeiçoar o uso dos recursos com de uma melhor distribuição das instâncias no tempo.

7.1. Trabalhos Futuros

Como trabalhos futuros, observamos a possibilidade de uma análise e uma avaliação mais detalhadas de como os acordos de nível de serviço podem determinar a seleção das estratégias de escalonamento e aprovisionamento tanto para aplicações em nuvem como para aplicações locais.

Existe também a necessidade da investigação de mecanismos que evitem que a sobrecarga no esforço de coordenação possam tornar o *Controller* um gargalo na escalabilidade de sistemas OddCI, especialmente quando manipularem redes de *broadcast* de larga escala. Ficou evidenciado que a concorrência pelo uso do canal de *broadcast*, notadamente em contextos que envolvam a coordenação de muitas instâncias 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.

Avaliar o desempenho do método de estimativa de PNA exploratório em comparação com o método informativo, assim como a comparação entre o método de seleção de PNA por ranqueamento em relação ao método por ordem de descoberta.

A possibilidade de avaliação do consumo de energia e recursos nos dispositivos utilizados nos sistemas OddCI, com o retorno para o contribuinte do recurso com um modelo de negócio que vise o abono dos seus gastos é outra possibilidade.

Referências

ABNT, 2009a. *Televisão digital terrestre - Codificação de dados e especificações de transmissão para radiodifusão digital - Parte 2. NBR 15606-2,* s.l.: s.n.

ABNT, 2009b. *Televisão digital terrestre - Codificação de dados e especificações de transmissão para radiodifusão digital - Parte 3. NBR 15606-3,* s.l.: s.n.

ABNT, 2009c. *Televisão digital terrestre – Codificação de dados e especificações de transmissão para radiodifusão digital - Parte 4. NBR 15606-4,* s.l.: s.n.

Anderson, D. P., 2004. BOINC: A System for Public-Resource Computing and Storage. *Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)*, pp. 4-10.

Anderson, D. P. et al., 2002. SETI@Home An Experiment in Public-Resource Computing. *Communications of the ACM Archive , vol. 45(11)*, pp. 56-61.

Andrade, N., Brasileiro, F., Cirne, W. & Mowbray, M., 2005. *Automatic Grid Assembly by Promoting Collaboration in Peer-to-Peer Grids". Submitted for Publication. HP Labs,* s.l.: s.n.

Andrzejak, A., Kondo, D. & Anderson, D. P., 2008. *Ensuring Collective Availability in Volatile Resource Pools Via Forecasting.* Berlin, Heidelberg, Springer-Verlag, pp. 149-161.

Aron, R. & Chana, I., 2012. Formal QoS Policy Based Grid Resource Provisioning Framework. *J. Grid Comput.*, #jun#, 10(2), pp. 249-264.

Batista, C. et al., 2007. TVGrid: A Grid Architecture to use the idle resources on a Digital TV network. *Proc. of 7th IEEE CCGRID´07*.

Blog, B. O., 2008. *Record 4.7 billion Television Viewers Watched Beijing Olympic Games 2008*, s.l.: s.n.

Braun, T. et al., 1998. *A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems.* s.l., s.n., pp. 330-335.

Chetty, M. & Buyya, R., 2002. Weaving Computational Grids: How Anal- ogous Are They with Electrical Grids?. *Computing in Science and Engineering (CiSE)*, pp. 61-71.

Cirne, W. et al., 2006. Labs of the World, Unite!!!. *Journal of Grid Computing*, Volume 4, pp. 225-246.

Cirne, W. et al., 2003. Running Bag-of-Tasks applications on computational grids: the MyGrid approach. *IEEE*.

Costa, R. et al., 2012a. OddCI-Ginga: Uma Plataforma para Computação de Alta Vazão baseada em Receptores de TV Digital. *SBRC*.

Costa, R. et al., 2012c. OddCI-Ginga: A Platform for High Throughput Computing Using Digital TV Receivers. *Grid2012*.

Costa, R., Brasileiro, F., Sousa, D. & Souza Filho, G. L., 2009. OddCI: on-demand distributed infrastruture. *MTAGS '09 Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers*.

Costa, R. et al., 2012b. Coordenação de Infraestruturas Computacionais Dinâmicas Baseadas em Redes de Broadcast. *X Workshop em Clouds, Grids e Aplicações*.

CSTB, NRC, AIP & NAS, 1989. *Supercomputers:Directions in Technology and Applications.* s.l.:The National Academies Press.

Datia, N., Moura-Pires, J., Cardoso, M. & Pita, H., 2005. *Temporal Patterns of TV watching for Portuguese Viewers.* s.l., s.n., pp. 151-158.

Dinda, P. A., 2006. Design, Implementation, and Performance of an Extensible Toolkit for Resource Prediction in Distributed Systems. *IEEE Trans. Parallel Distrib. Syst.*, #feb#, 17(2), pp. 160-173.

Dinda, P. & O'Hallaron, D., 1999. *An Extensible Toolkit for Resource Prediction in Distributed Systems,* s.l.: s.n.

Foster, I., 2002. *What is the Grid? A Three Point Checklist.* [Online]
Available at: http://www.mcs.anl.gov/~itf/Articles/WhatIsTheGrid.pdf
[Acesso em 13 Julho 2012].

Foster, I., Kesselman, C. & Tuecke, S., 2001. The Anatomy of the Grid Enabling Scalable Virtual Organizations. *Intl J. Supercomputer Applications*.

Godfrey, B., 2002. *A primer on distributed computing.* [Online]
Available at: http://www.bacchae.co.uk/docs/dist.html
[Acesso em 01 Julho 2012].

Gonzalez, T. & Sahni, S., 1978. Flowshop and Jobshop Schedules: Complexity and Approximation. *Operations Research Society of America*, Volume 26.

Hanzich, M. et al., 2011. On/off-line prediction applied to job scheduling on non-dedicated NOWs. *J. Comput. Sci. Technol.*, #jan#, 26(1), pp. 99-116.

Jain, R., 1991. *The Art of Computer Systems Performance Analysis.* s.l.:John Wiley and Sons.

Kacsuk, P., 2011. *How to Make BOINC-Based Desktop Grids Even More Popular?.* s.l., s.n., pp. 1871-1877.

Keahey, K., Doering, K. & Foster, I., 2004. *From Sandbox to Playground: Dynamic Virtual Environments in the Grid.* Washington, DC, USA, IEEE Computer Society, pp. 34-42.

Leite, L. E. C. & Souza Filho, G. L., 2005. *Sistema Brasilerio de Televisão Digital: Recomendações para o Modelo de Referência,* s.l.: s.n.

Lili, S., Shoubao, Y., Liangmin, G. & Bin, W., 2009. *A Markov Chain Based Resource Prediction in Computational Grid.* s.l., s.n., pp. 119-124.

Litzkow, M., Livny, M. & Mutka, M., 1988. Condor - A Hunter of Idle Workstations. *Proceedings of the 8th International Conference of Distributed Computing Systems*, June, pp. 104-111.

Lü, X., 2007. *Modeling of television audience viewing behavior.* [Online]
Available at: http://hdl.handle.net/2031/5040
[Acesso em 31 julho 2012].

Marupally, P. R. & Paruchuri, V., 2010. Comparative Analysis and Evaluation of Botnet Command and Control Models. *Advanced Information Networking and Applications, International Conference on,* Volume 0, pp. 82-89.

McLaughlin, L., 2004. Bot software spreads, causes new worries. *Distributed Systems Online, IEEE,* , 5(6), p. 1.

Mickens, J. W. & Noble, B. D., 2006. Improving distributed system performance using machine availability prediction. *SIGMETRICS Perform. Eval. Rev.*, #sep#, 34(2), pp. 16-18.

Németh, C., Dózsa, G., Lovas, R. & Kacsuk, P., 2004. The P-GRADE Grid Portal. In:: A. Laganá, et al. eds. *Computational Science and Its Applications – ICCSA 2004.* s.l.:Springer Berlin / Heidelberg, pp. 10-19.

Pande lab, S. U., 2011. *PS3 FAQ.* [Online]
Available at: http://folding.stanford.edu/English/FAQ-PS3
[Acesso em 1 Julho 2012].

Pande Lab, S. U., 2012. *Folding@home Distributed Computing.* [Online]
Available at: http://fah-web.stanford.edu/cgi-bin/main.py?qtype=osstats
[Acesso em 12 Julho 2012].

Raicu, I., Foster, I. T. & Zhao, Y., 2008. Many-Task Computing for Grids and Supercomputers. *IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS08).*

Ren, X., Lee, S., Eigenmann, R. & Bagchi, S., 2007. Prediction of Resource Availability in Fine-Grained Cycle Sharing Systems Empirical Evaluation. *Journal of Grid Computing,* Volume 5, pp. 173-195.

Rood, B. & Lewis, M., 2009. Grid Resource Availability Prediction-Based Scheduling and Task Replication. *Journal of Grid Computing,* Volume 7, pp. 479-500.

Ross, P. E., 2008. *Why CPU Frequency Stalled.* [Online]
Available at: http://spectrum.ieee.org/computing/hardware/why-cpu-frequency-stalled
[Acesso em novembro 2012].

Sadashiv, . N. & Kumar, S. M. D., 2011. Cluster, Grid and Cloud Computing: A Detailed Comparison. *The 6th International Conference on Computer Science & Education*, pp. 477-482.

Sánchez, I. et al., 2009. *Botnet-inspired architecture for interactive spaces.* New York, NY, USA, ACM, pp. 10:1--10:10.

Science, C. et al., 1989. *Supercomputers:Directions in Technology and Applications.* s.l.:The National Academies Press.

Soares, L. F. G., 2006. *MAESTRO: The Declarative Middleware Proposal for the SBTVD,* s.l.: s.n.

Sutter, H., 2005. The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software. *Dr. Dobb's Journal,* 30(3), pp. 202-210.

Tecmundo, 2012. *Afinal, como é medido o IBOPE da TV e internet?.* [Online]
Available at: http://www.tecmundo.com.br/televisao/18855-afinal-como-e-medido-o-ibope-da-tv-e-internet-.htm
[Acesso em 31 julho 2012].

Thain, D., Tannenbaum, T. & Livny, M., 2006. How to measure a large open-source distributed system. *Concurrency and Computation: Practice and Experience,* 18(15), pp. 1989-2019.

Toth, D., Mayer, R. & Nichols, W., 2011. *Increasing Participation in Volunteer Computing.* s.l., s.n., pp. 1878-1882.

Ullman, J. D., 1975. NP-Complete Scheduling Problems. *Journal of Computer and System Sciences*, Volume 10, pp. 384-393.

UOL, 2013. *"Salve Jorge" tem 38 pontos de média no Ibope e bate recorde.* [Online]
Available at: http://televisao.uol.com.br/noticias/redacao/2013/01/28/salve-jorge-tem-38-pontos-de-media-no-ibope-e-bate-recorde.htm

Varga, A. & Hornig, R., 2008. An overview of the omnet++ simulation. *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops (Simutools '08)*.

Warneke, D. & Kao, O., 2009. *Nephele: efficient parallel data processing in the cloud..* s.l., ACM.

Wolski, R., 1998. Dynamically forecasting network performance using the Network Weather Service. *Cluster Computing*, #jan#, 1(1), pp. 119-132.

Wolski, R., Spring, N. T. & Hayes, J., 1999. The network weather service: a distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(5–6), pp. 757-768.