GUILHERME GALANTE

EXPLORANDO A ELASTICIDADE EM NÍVEL DE PROGRAMAÇÃO NO DESENVOLVIMENTO E NA EXECUÇÃO DE APLICAÇÕES CIENTÍFICAS

Tese apresentada como requisito parcial à obtenção do grau de Doutor. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Luis Carlos Erpen de Bona

CURITIBA

2014

G146e Galante, Guilherme
Explorando a elasticidade em nível de programação no desenvolvimento e na execução de aplicações científicas / Guilherme Galante. – Curitiba, 2014.
115f. : il.; tab.

Tese (doutorado) - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-graduação em Informática, 2014.
Orientador: Luis Carlos Erpen de Bona
Bibliografia: p. 95-106.

1. Computação em nuvem. 2. Framework (Programa de computador). 3. Elasticidade. I. Bona , Luis Carlos Erpen de. II. Universidade Federal do Paraná. III. Título.

CDD: 006.67

Ministério da Educação
Universidade Federal do Paraná
Programa de Pós-Graduação em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa do aluno de Doutorado em Ciência da Computação, Guilherme Galante, avaliamos a tese de doutorado intitulada *"Explorando a elasticidade em nível de programação no desenvolvimento e a execução de aplicações científicas"*, cuja defesa pública foi realizada no dia 28 de abril de 2014, às 09:30 horas, no Departamento de Informática do Setor de Ciências Exatas da Universidade Federal do Paraná. Após avaliação, decidimos pela: (X) **aprovação** do candidato. ( ) **reprovação** do candidato.

Curitiba, 28 de abril de 2014.

Prof. Dr. Luis Carlos Erpen de Bona
DINF/UFPR – Orientador

Prof. Dr. Bruno Schulze
LNCC – Membro Externo

Prof. Dr. Luiz Fernando Bittencourt
UNICAMP – Membro Externo

Prof. Dr. Carlos Alberto Maziero
UTFPR – Membro Interno

Prof. Dr. Fabiano Silva
DINF/UFPR – Membro Interno

MINISTÉRIO DA EDUCAÇÃO - UNIVERSIDADE FEDERAL DO PARANÁ - Setor de Ciências Exatas - Coordenação de Pós-Graduação em Informática

Pouco conhecimento faz com que as pessoas se sintam orgulhosas. Muito conhecimento, que se sintam humildes. É assim que as espigas sem grãos erguem desdenhosamente a cabeça para o céu, enquanto que as cheias as baixam para a terra, sua mãe.

Leonardo Da Vinci

AGRADECIMENTOS

Quatro anos depois, quilos a mais e cabelos de menos, gostaria de agradecer todos aqueles que estiveram presentes e colaboraram de alguma forma com o desenvolvimento deste trabalho.

Em primeiro lugar, gostaria de agradecer à minha esposa Juliana pelo amor, apoio e compreensão nestes últimos anos. Agora prometo que não fico mais longe de casa!

Agradeço também aos meus pais, Luiza e Sinval, e às minhas irmãs Patricia e Camylla, por terem sempre me motivado e incentivado ao longo destes anos. Um agradecimento especial para a tia Vanda pelo suporte dado em Curitiba e por ter me acolhido tão bem em sua casa.

Obrigado ao meu orientador, Prof. Luis Carlos Erpen De Bona, pela confiança depositada ao me aceitar no Programa de Pós-Graduação e pelas experiências compartilhadas ao longo dos anos. Valeu, Bona! Agradecimentos também aos demais professores e funcionários da UFPR que, de alguma forma, contribuíram para a minha formação pessoal e para construção deste trabalho.

Aos meus amigos do Larsis, em especial, Luiz Antonio, Ailton, Edmar e Maurício, pelas discussões de ideias, almoços, cafés, churrascos, jogos de loteria (nada ainda?!), o meu muito obrigado. Sei que ao final da jornada cada um toma o seu rumo, mas espero que continuemos em contato.

Obrigado também aos meus amigos e colegas do Colegiado de Ciência da Computação da Unioeste pelo incentivo e pelas dicas de como levar um doutorado.

Por fim, um agradecimento aos meus amigos e quase irmãos, André e Edmar, pela amizade, pelas conversas e pelos bons momentos de descontração.

Muito obrigado!

SUMÁRIO

LISTA DE ABREVIATURAS

API	*Application Programming Interface*
ARMA	*Autoregressive Moving Average Models*
ASG	*Auto-Scale Group*
CEO	*Chief Executive Officer*
CERN	*Conseil Européen pour la Recherche Nucléaire*
CPU	*Central Processing Unit*
CRM	*Customer Relationship Management*
CUDA	*Compute Unified Device Architecture*
DOE	*Department of Energy*
E/S	Entrada e Saída
EC2	*Elastic Compute Cloud*
EMBL	*European Molecular Biology Laboratory*
ESA	*European Space Agency*
FTP	*File Transfer Protocol*
GB	Gigabyte
GPU	*Graphical Processing Unit*
HPC	*High-Performance Computing*
HTC	*High-Throughput Computing*
IaaS	*Infrastructure-as-a-Service*
IBM	*International Business Machines*
IEEE	*Institute of Electrical and Electronics Engineers*
MB	Megabyte
MOP	*Memory Overprovisioning Percentage*
MPI	*Message Passing Interface*
MV	Máquina Virtual
MTC	*Many-Task Computing*
NASA	*National Aeronautics and Space Administration*
NSF	*National Science Foundation*
OLAM	*Ocean-Land-Atmosphere Model*
OMR	*Online Mesh Refinement*
OpenMP	*Open Multi-Processing*
PaaS	*Platform-as-a-Service*
PC	*Personal Computer*
PPA	Percentual de Provisionamento Adicional
S3	*Simple Storage Service*
SaaS	*Software-as-a-Service*
SCP	*Secure Copy*

SGI	*Silicon Graphics, Inc.*
SLA	*Service Level Agreement*
SLO	*Service Level Objective*
SOA	*Service Oriented Architecture*
SSH	*Secure Shell*
RAM	*Random Access Memory*
TB	Terabyte
VCPU	*Virtual Central Processing Unit*
XML	*eXtensible Markup Language*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

A elasticidade pode ser definida como a capacidade de um sistema de modificar dinamicamente os recursos computacionais utilizados por uma aplicação. Embora diversos mecanismos de elasticidade tenham sido propostos, ainda apresentam uma série de limitações ao fornecer suporte à elasticidade para aplicações científicas. Neste trabalho, propõe-se uma abordagem para o desenvolvimento de aplicações científicas elásticas, na qual o controle da elasticidade é feito em nível de programação. Isso significa que o controle de elasticidade é incorporado ao código-fonte, permitindo que as ações de alocação e desalocação de recursos possam ser realizadas pela própria aplicação e não dependa de mecanismos externos ou interação com o usuário. A construção de aplicações elásticas de acordo com abordagem proposta baseia-se no conceito de primitivas de elasticidade, um conjunto de funções que permitem que as aplicações comuniquem-se com a nuvem para solicitar ou liberar recursos, bem como para coletar informações do ambiente virtual e da nuvem. Assim, é possível desenvolver controladores de elasticidade sob medida que permitem que aplicações ajustem seus próprios recursos de acordo com suas demandas ou de modo a satisfazer algum critério específico, por exemplo, custo ou desempenho. A abordagem também permite que bibliotecas e *frameworks* de programação paralela possam ser construídas ou adaptadas de modo a oferecer elasticidade de modo transparente. Para permitir a construção de aplicações elásticas utilizando a abordagem proposta, desenvolveu-se o *framework* Cloudine. O Cloudine fornece as primitivas de elasticidade e um ambiente de execução, o qual oferece o suporte para a execução das aplicações elásticas na nuvem. A exploração da elasticidade em nível de programação é validada por um conjunto de experimentos realizados utilizando o Cloudine. O *framework* é utilizado com sucesso para fornecer elasticidade a um conjunto de aplicações, dentre as quais destacam-se uma aplicação de montagem de genomas (SAND) e um modelo climático (OLAM). O Cloudine também é usado para estender a biblioteca OpenMP do GCC (*libgomp*) para oferecer elasticidade de modo automático e transparente.

Palavras-Chave: Computação em Nuvem, Aplicações Científicas, Elasticidade, Cloudine.

ABSTRACT

Elasticity is defined as the ability to adaptively scale resources up and down in order to meet varying application demands. Although several mechanisms to provide this feature are offered by public cloud providers and in some academic works, we argue that these solutions present limitations in providing elasticity for scientific applications, since they are not developed to this purpose and cannot consider the particularities of this class of applications. In this thesis we propose an approach for exploring the elasticity in scientific applications, in which the elasticity control is embedded within application code and the elasticity actions (allocation and deallocation of resources) are performed by the application itself, based in its runtime requirements or internal events. The development of embedded elasticity controllers is based on the concept of elasticity primitives, which are basic functions that allow to perform requests for allocation or deallocation of resources directly to the cloud. Thus, it is possible to develop tailor made elasticity controllers that enable applications to adjust its own resources according to its demands or to satisfy some specific criteria, such as cost or performance. It is also possible to develop elasticity-aware parallel processing middleware that transparently support applications elasticity. To enable the construction of elastic applications using the presented approach, we developed the Cloudine framework. Cloudine provides the primitive set and a runtime environment, which supports the execution of elastic application in the cloud. The proposed approach is validated by a set of experiments using Cloudine. The framework was successfully used to provide elasticity to a number of applications, among which we highlight a genome assembler (SAND) and a climate model (OLAM). The Cloudine is also used to extend the GCC's OpenMP library (*libgomp*) to provide automatic allocation of resources.

Keywords: Cloud Computing, Scientific Applications, Elasticity, Cloudine.

CAPÍTULO 1

INTRODUÇÃO

O uso da computação em atividades científicas das mais diversas áreas é cada vez mais comum, tanto que é considerado por alguns autores como o terceiro pilar das ciências, juntamente com a teoria e a experimentação. A evolução de plataformas e de infraestruturas de computação científica tem possibilitado a realização de experimentos que seriam impossíveis de se realizar há apenas 10 anos, mudando a forma como os cientistas fazem ciência [54].

Como resultado direto do uso de computadores, muitas áreas de pesquisa têm apresentado progressos de forma nunca vista. É importante notar, no entanto, que esse progresso acaba atribuindo exigências adicionais sobre o processo de automação, fazendo-se necessária uma quantidade cada vez maior de recursos de processamento, armazenamento e comunicação para a produção de resultados para problemas de tamanho cada vez maior [122].

Recentemente, a computação em nuvem surgiu como uma alternativa para o fornecimento de recursos computacionais para a execução de aplicações científicas. A computação em nuvem refere-se a um modelo de computação que oferece acesso imediato a um conjunto compartilhado de recursos de computação que podem ser facilmente configurados e provisionados conforme a demanda do usuário. O compartilhamento dos recursos é garantido pelo uso de virtualização, que permite que vários ambientes virtuais sejam executados sobre o hardware físico, permitindo a otimização do uso de recursos e a economia de escala [101].

O uso da virtualização permite ainda que os recursos da nuvem sejam fornecidos de modo escalável, dinâmico e de acordo com as demandas do usuário. Isso significa que os recursos que compõem o ambiente computacional podem ser adicionados ou removidos, a qualquer momento e em qualquer quantidade, sem causar a interrupção dos serviços [28]. Para o cliente, os recursos computacionais disponíveis são quase que infinitos, sendo da responsabilidade do provedor da nuvem ter recursos suficientes para satisfazer as necessidades de todos os seus clientes [53]. Essa flexibilidade na alocação de recursos é chamada de *elasticidade*.

Do ponto de vista das aplicações científicas, a elasticidade pode ser usada para encontrar a quantidade de recursos mais apropriada para a execução de aplicações cujos requisitos não podem ser determinados antecipadamente com exatidão devido a mudanças nas cargas de trabalho, ou ainda, devido a mudanças na estrutura da aplicação [83]. Dessa forma, uma aplicação pode começar com uma quantidade mínima de recursos e, durante sua execução, novos recursos podem ser requisitados. Isso permite que recursos sejam alocados, por exemplo, para tratar uma demanda específica, para melhorar o desempenho da aplicação, ou para tirar proveito da disponibilidade de recursos ociosos ou de baixo custo [68].

Atualmente, vários mecanismos de elasticidade são oferecidos por provedores de nuvem pública, como o Amazon EC2, o GoGrid e o Rackspace [66]. Estes mecanismos têm sido tradicionalmente empregados para escalar aplicações cliente-servidor (servidores web, e-mail e banco de dados, por exemplo) com o intuito de evitar as questões relacionadas com o provisionamento excessivo ou insuficiente de recursos [45].

Em geral, a elasticidade oferecida nestes mecanismos baseia-se na variação do número de máquinas virtuais empregadas pelos componentes da aplicação e no uso de balanceadores de carga para dividir a carga de trabalho entre os diversos servidores virtualizados. O controle da elasticidade é realizado por meio de um conjunto de regras e condições para decidir sobre a adição ou remoção de recursos. Essa regras baseiam-se em informações provenientes de um sistema de monitoramento que coleta dados sobre a carga de trabalho (número de clientes, conexões, etc) e sobre a utilização de recursos das máquinas virtuais [139].

Embora sejam utilizados com sucesso em aplicações cliente-servidor, tais mecanismos ainda apresentam uma série de limitações para fornecer elasticidade para aplicações científicas [151, 68]: (1) A simples adição de máquinas virtuais e o uso de balanceadores de carga são inefetivos para grande parte das aplicações científicas, uma vez que estas exigem um maior grau de coordenação e sincronização. Além disso, as aplicações não são desenvolvidas de modo a detectar e utilizar os recursos adicionados. (2) As cargas de trabalho de aplicações científicas geralmente são definidas por arquivos de entrada e parâmetros de configuração em vez de requisições externas. Isso torna difícil estimar se recursos devem ou não ser alocados, considerando que não há informações de cargas externas. (3) O consumo de recursos em aplicações científicas é di-

ferente do apresentada em aplicações baseadas no servidor: Enquanto o primeiro tipo tende a consumir todos os recursos atribuídos, independentemente do montante provisionado, o último consome os recursos de acordo com a variação de sua carga de trabalho. Considerando que os mecanismos não conseguem coletar informações sobre os eventos gerados internamente pela aplicação, é difícil estimar se a adição ou remoção de recursos são realmente necessários.

Deve-se considerar ainda que existem vários modelos de aplicações, cada um com seus próprios detalhes de implementação e comportamentos de execução. Assim, para estimar com precisão as demandas das aplicações, seriam necessários controladores especializados para cada modelo. Neste sentido, alguns trabalhos acadêmicos têm desenvolvido soluções que permitem o desenvolvimento de aplicações científicas elásticas para classes específicas de aplicações. É possível encontrar trabalhos com foco em *workflows* [37], MapReduce [46, 80], MPI (*Message Passing Interface*) [118] e aplicações mestre-escravo [116].

Considerando o exposto, este trabalho tem como objetivo propor uma abordagem adequada para o desenvolvimento de aplicações científicas elásticas, livre das limitações presentes nas soluções de elasticidade atuais. Nesta abordagem, o controle da elasticidade é definido em *nível de programação*, o que significa que toda lógica de controle, antes realizada por um mecanismo externo, é incorporada ao código-fonte da aplicação. Assim, torna-se possível desenvolver aplicações elásticas capazes de gerenciar de modo adequado a alocação de seus próprios recursos, uma vez que o controle da elasticidade faz parte de sua lógica e que os elementos internos da aplicação passam a ser considerados.

Neste trabalho, a construção de aplicações elásticas de acordo com abordagem proposta baseia-se no conceito de *primitivas de elasticidade*. Estas primitivas correspondem a um conjunto de funções básicas que permitem que as aplicações comuniquem-se com a infraestrutura da nuvem subjacente para solicitar ou liberar recursos, bem como para coletar informações do ambiente virtual e da nuvem. A partir destas primitivas, soluções de elasticidade sob medida para cada aplicação podem ser desenvolvidas.

Essa abordagem dá origem a um novo paradigma para o projeto e o desenvolvimento de aplicações, no qual os recursos passam a ser tratados como elementos variáveis de um programa, podendo ser instanciados e modificados ao longo da execução. Isso permite que funcionalidades

não previstas nas soluções atuais de elasticidade possam ser agregadas às aplicações científicas.

É possível desenvolver aplicações elásticas capazes de adaptar o seu próprio ambiente de execução de acordo com suas demandas ou de modo a adaptá-lo a cenários específicos, relacionados a fatores como custo e desempenho. Pode-se também modificar bibliotecas e *frameworks* de programação paralela de modo que ofereçam suporte à elasticidade e permitam que aplicações elásticas sejam construídas de modo transparente a partir delas.

Para oferecer suporte para a construção e a execução de aplicações elásticas utilizando a abordagem proposta, desenvolveu-se o *framework* Cloudine. O *framework* compreende dois componentes principais: o Ambiente de Execução, que gerencia o provisionamento de recursos utilizando a infraestrutura de nuvens e, a API de Elasticidade, que fornece o conjunto de primitivas para habilitar a interação da aplicação com a infraestrutura da nuvem.

A exploração da elasticidade em nível de programação é validada por um conjunto de experimentos realizados utilizando o Cloudine. Nestes experimentos, o *framework* é utilizado com sucesso para adicionar suporte à elasticidade a aplicações sintéticas e também a duas aplicações reais, mais especificamente uma aplicação de montagem de genomas (SAND) e um modelo atmosférico (OLAM). O Cloudine também é empregado no desenvolvimento de uma versão elástica da API OpenMP do GCC (*libgomp*), na qual a alocação de recursos é feita de modo automático e transparente pelas próprias diretivas (*pragmas*) do OpenMP.

O restante do documento está organizado da seguinte maneira. O Capítulo 2 apresenta uma visão geral da computação em nuvem, abordando os principais conceitos envolvidos neste modelo de computação. No Capítulo 3 aborda-se o uso da computação em nuvem para a realização de experimentos científicos. O Capítulo 4 descreve o estado-da-arte da exploração da elasticidade em nuvens computacionais. No Capítulo 5, aborda-se o uso da elasticidade em nível de programação na construção de aplicações científicas elásticas e apresenta-se o *framework* Cloudine. No Capítulo 6 apresentam-se os testes e os resultados obtidos com o uso do Cloudine na construção aplicações elásticas. No Capítulo 7 apresenta-se a versão elástica da API OpenMP e seus respectivos resultados. Por fim, o Capítulo 8 conclui este trabalho.

CAPÍTULO 2

COMPUTAÇÃO EM NUVEM: VISÃO GERAL

O conceito atual de Computação em Nuvem (*Cloud Computing*) começou a tomar forma em 2006, quando Eric Schmidt, CEO da Google, utilizou pela primeira vez o termo para descrever o modelo de negócios de sua empresa [127, 107]. No mesmo ano, a Amazon usou a denominação para lançar seu serviço *Elastic Compute Cloud* (EC2) [107] como parte do Amazon Web Services. Atualmente existem muitas definições para computação em nuvem na literatura, sendo que nenhuma é considerada unânime e definitiva, o que mostra que o conceito ainda continua em desenvolvimento.

No trabalho de Vaquero [140] são avaliadas mais de 20 definições que serviram de base para uma proposta unificada. Segundo o autor, uma nuvem computacional pode ser tratada, em tradução livre, como:

> "*um grande conjunto de recursos virtualizados e compartilhados (hardware, plataformas de desenvolvimento ou software) que podem ser facilmente acessados e utilizados. Esses recursos podem ser dinamicamente reconfigurados para se ajustarem a uma carga variável de trabalho, permitindo uso otimizado dos mesmos. Este conjunto de recursos é tipicamente explorado por um modelo de pagamento por utilização (pay-per-use) em que as garantias são oferecidas pelo provedor de infraestrutura por meio de contratos de serviço personalizados (Service Level Agreement – SLA).*"

Baseado nessa e em outras definições encontradas na literatura [36, 65, 140] é possível identificar a computação em nuvem como modelo de computação e também como modelo de negócios. Apesar de recentes, ambos os modelos possuem características em comum com diversas outras tecnologias consolidadas, tais como a virtualização, a computação utilitária, a computação em grade, entre outras.

De acordo com Mell et al. [101], o modelo computacional de nuvem tem como base o uso de uma infraestrutura composta por um conjunto (*pool*) de servidores e de toda a infraestrutura necessária para seu funcionamento (eletricidade, refrigeração, rede de interconexão e armazenamento), que são compartilhados entre inúmeros usuários (*multi-tenancy*). O compartilhamento é feito através do uso de virtualização, que permite que vários ambientes virtuais sejam executados sobre o hardware físico, permitindo a otimização do uso de recursos e a economia de escala. O acesso aos recursos fornecidos é feito remotamente por meio de uma rede ou Internet.

O uso da virtualização permite ainda que os recursos da nuvem sejam fornecidos de modo escalável, elástico e de acordo com as demandas do usuário. Isso significa que os recursos da nuvem podem ser alocados pelo usuário de modo automático, a qualquer momento e em qualquer quantidade. Do ponto de vista do cliente, os recursos computacionais disponíveis são quase que infinitos, sendo da responsabilidade do provedor da nuvem ter recursos suficientes para satisfazer as necessidades de todos os seus clientes [53].

Essa facilidade de requisitar ou liberar recursos de forma elástica torna as nuvens uma infraestrutura adequada para a execução de aplicações dinâmicas, ou seja, aquelas cujas demandas podem ser alteradas durante a sua execução, quer devido a mudanças nos requisitos em tempo de execução, quer devido a mudanças na estrutura da aplicação [83].

Por sua vez, o modelo nuvem de negócio define os papéis e suas contrapartidas no fornecimento e aquisição dos recursos computacionais. Neste contexto surgem dois tipos de atores: usuários e provedores. Em linhas gerais, usuários são aqueles que se utilizam dos recursos, devendo ou não pagar pelo uso dos mesmos, e os provedores são os responsáveis pela adequada implantação e operação da infraestrutura e dos mecanismos que permitem que eles ofereçam serviços. Esta relação entre os usuários e a organização que mantém o sistema de computação define os quatro modelos de implantação de nuvens computacionais [101]: *privada*, *pública*, *comunitária* e *híbrida*.

Pode-se ainda classificar um ambiente de nuvem de acordo com o seu modelo de serviço, ou seja, pelo tipo de recurso que oferece. Basicamente, são definidos três modelos [28]: Infraestrutura como Serviço (*Infrastructure as a Service* – IaaS), Plataforma como Serviço (*Platform as a Service* – PaaS) e Software como Serviço (*Software as a Service* – SaaS).

De modo a oferecer uma visão geral sobre o modelo de computação em nuvem, a Seção 2.1 apresenta as principais tecnologias e conceitos envolvidos no surgimento do conceito de nuvem, a Seção 2.2 aborda com mais detalhes os quatro modelos de implantação e, finalmente, a Seção 2.3 descreve os três modelos de serviço.

2.1 Conceitos e tecnologias precedentes

A computação em nuvem pode ser vista como um paradigma de computação recente, mas que tem origens em um conjunto de tecnologias e conceitos já existentes. Alguns deles foram vistos como modismos em seus estágios iniciais de desenvolvimento, porém, mais tarde receberam atenção significativa da academia e da indústria, alcançando a sua maturidade.

As primeiras aspirações do modelo de computação em nuvem remontam aos anos 60, quando o cientista da computação John McCarthy previu que a computação poderia ser um dia organizada como uma utilidade pública, onde um serviço é prestado por meio de uma assinatura ou pagamento [107]. Em 1966, Douglas Parkhill publica o livro *The Challenge of the Computer Utility* (O Desafio da Computação Utilitária, em tradução livre) [112], no qual o autor explora muitas das características modernas da computação em nuvem (provisionamento elástico, pagamento pelo uso, entre outras), bem como seus modelos de implantação e prováveis usos. A influência dos conceitos oriundos da computação utilitária é evidente na construção do modelo de negócio da computação em nuvem.

Outra tecnologia relacionada ao modelo de nuvem e também originada a partir das ideias da computação utilitária é a computação em grade (*grid computing*). O termo, cunhado no final da década de 1990, deriva-se de uma analogia com a rede elétrica (*power grid*) e reflete o objetivo de tornar o uso de recursos computacionais distribuídos tão simples quanto ligar um aparelho na rede elétrica [62], embora ainda não tenha alcançado este nível de abstração.

Em uma definição atual, uma grade pode ser vista como uma infraestrutura de larga escala geograficamente distribuída composta por recursos heterogêneos interligados por rede, mantidos por múltiplas organizações e coordenados para fornecer suporte à computação de modo confiável a uma ampla gama de aplicações [32]. O desenvolvimento das grades se deu com maior ênfase no meio acadêmico com o intuito de juntar recursos disponíveis em domínios

diferentes (por exemplo, universidades ou grupos de pesquisa) de modo a fornecer o poder computacional demandado pelas aplicações científicas.

A computação em nuvem é semelhante à computação em grade na medida em que também agrega recursos distribuídos para oferecer uma grande capacidade de computação, sendo muitas vezes vista como uma evolução ou subconjunto das grades computacionais [27, 149]. Contudo, a evolução que a computação em nuvem representa está na mudança do foco de uma infraestrutura que fornece apenas recursos computacionais (no caso das grades) para uma infraestrutura com controle centralizado baseada em economia de escala, que se vale de tecnologias de virtualização em vários níveis (hardware, plataforma e aplicações) para otimizar o uso da infraestrutura dos centros de dados.

O uso da virtualização para estes fins também não é recente. As máquinas virtuais foram originalmente desenvolvidas, entre os anos 60 e 70, para fornecer ambientes dedicados para cada usuário no mainframe VM/370 da IBM [90]. Por meio da virtualização pode-se criar inúmeras máquinas virtuais em um mesmo hardware físico, encapsulando um computador completo, incluindo CPU, memória, sistema operacional e rede.

Outro aspecto importante é que uma máquina virtual pode ser personalizada, podendo executar seu próprio sistema operacional e aplicações, independentemente das camadas subjacentes. Essa abstração fornecida pela virtualização é fundamental para o conceito de computação em nuvem, principalmente para o modelo IaaS, já que é por meio dela que são garantidas as características de compartilhamento, economia de escala, elasticidade e escalabilidade da nuvem [150].

Considerando uma nuvem como um ambiente dinâmico, elástico, com carga de trabalho altamente variável e com uma grande quantidade e variedade de recursos compartilhados, são necessários mecanismos adequados para o gerenciamento de toda a infraestrutura. Muitas vezes os níveis de complexidade podem exceder a capacidade humana, sendo necessária a automatização deste processo [134]. Nesse sentido, os conceitos relacionados à Computação Autônoma (*Autonomic Computing*) podem ser aplicados para fornecer tais mecanismos de gerência e controle.

O termo Computação Autônoma foi definido em 2001 pela IBM e refere-se à construção de sistemas computacionais capazes de se auto-gerenciarem com a mínima interferência humana [86]. Com a utilização de sistemas autônomos, há menos necessidade de manutenção do sistema, uma vez que as várias tarefas do sistema, tais como segurança ou configuração, são mantidas automaticamente por meio de verificações e diagnósticos. Dessa forma, as nuvens podem incorporar elementos da computação autônoma na realização de diversas tarefas, tais como, gerenciamento da capacidade da infraestrutura, provisionamento de máquinas virtuais, balanceamento da carga de trabalho, recuperação de falhas e controle de execução de aplicações [155].

Se a virtualização e a computação autônoma são conceitos indispensáveis para os serviços de Infraestrutura (IaaS), o conceito de Arquitetura Orientada em Serviço (*Service-Oriented Architecture* – SOA) são vitais para os serviços de software da nuvem (PaaS e SaaS) [146]. SOA é um estilo de arquitetura de software cujo princípio fundamental determina que as funcionalidades implementadas pelas aplicações devem ser disponibilizadas na forma de serviços. Diversos serviços podem ser combinados para a criação novas aplicações, chamadas de *mashups*.

Atualmente, muitos serviços estão disponíveis no mercado, tais como os fornecidos pelas APIs oferecidas pela Google [9], Amazon [1], Twitter [20] entre outras.

A Tabela 2.1 resume as contribuições das tecnologias para a área da computação em nuvem. Como pode-se ver de maneira sintetizada na tabela, a computação em nuvem está relacionada com estas tecnologias de maneira indissociável, tanto no sentido técnico da implementação da nuvem, quanto no seu modelo de negócio.

2.2 Modelos de implantação

Independentemente da classe de serviço prestado, uma nuvem pode ser classificada quanto ao seu modelo de implantação como pública, privada, comunitária ou híbrida [101]. Para caracterizar tais modelos, analisa-se quatro atributos: (1) quem é o proprietário da infraestrutura; (2) quem gerencia a infraestrutura; (3) onde a infraestrutura está alocada; e (4) quem acessa os serviços da nuvem.

Tabela 2.1: Tecnologias e características atribuídas à computação em nuvem

Tecnologia	**Características**
Computação Utilitária	• modelo de negócios (*pay-per-use*) • recursos computacionais como serviço
Computação em Grade	• larga escala • compartilhamento de recursos • acesso de recursos pela Internet • computação sob demanda
Virtualização	• otimização do uso de recursos • economia de escala • elasticidade • personalização dos ambientes • escalabilidade • modelo IaaS
Computação Autônoma	• gerência automatizada
SOA	• implementação de software nos modelos SaaS e PaaS

O modelo público de nuvem baseia-se na oferta de recursos em escala massiva ao público, podendo ou não cobrar pelo seu uso. Toda a infraestrutura é mantida e gerenciada pelo próprio provedor. Os usuários das nuvens públicas são considerados não confiáveis, o que significa que não estão ligados à organização como empregados e que o usuário não tem acordos contratuais com o provedor, exceto o de consumidor/fornecedor. Neste caso, as garantias de qualidade de serviço são explicitadas em contratos de nível de serviço (SLA - *Service Level Agreement*) estabelecidos entre o usuário e o provedor. Os termos destes contratos definem as obrigações legais do provedor, embora verificar o seu efetivo cumprimento possa ser bastante difícil [28].

Por sua vez, uma nuvem privada é implantada para o uso exclusivo da própria organização, oferecendo serviços dentro de sua própria rede corporativa (ou sobre uma rede privada virtual) e os usuários são tipicamente seus próprios funcionários.

Uma alternativa introduzida recentemente, chamada de *nuvem privada virtual*, mescla características de nuvens privadas e públicas. Essa solução consiste em uma nuvem privada alo-

cada na infraestrutura física de uma nuvem pública. Neste caso, assegura-se que o armazenamento de dados e o processamento é feito apenas em servidores dedicados, ou seja, esses servidores não são compartilhados com qualquer outro cliente do provedor da nuvem, evitando alguns problemas gerados pelo compartilhamento e co-alocação [107, 53].

O modelo de implantação comunitário deriva-se da computação em grade e da computação voluntária, na qual organizações com características, necessidades e interesses semelhantes compartilham recursos comuns. Cada participante pode fornecer recursos, consumir recursos ou ambos. Para que a nuvem seja funcional, é necessário que pelo menos um dos membros faça o papel de provedor [147, 26].

Os principais benefícios do uso de nuvens comunitárias são o compartilhamento dos recursos entre as distintas organizações, a possibilidade de unir poder computacional distribuído entre as organizações, melhora do uso das infraestruturas instaladas e a facilidade de acesso e distribuição de dados e informações de interesse coletivo [28].

Por fim, uma nuvem híbrida é uma composição de duas ou mais nuvens de tipos diferentes (comunitárias, privadas ou públicas) que permanecem entidades únicas, mas são unidas de alguma maneira, oferecendo os benefícios da implantação de modelos múltiplos. Isso permite, por exemplo, o acesso a recursos adicionais, caso os recursos da nuvem privada não sejam suficientes para picos de carga de trabalho, ou ainda oferecer tipos de recursos disponíveis na nuvem pública que não estão disponíveis na nuvem privada [101].

A Tabela 2.2 conclui a seção comparando as principais características dos modelos de implantação de nuvens computacionais.

Tabela 2.2: Modelos de implantação de nuvens

	Gerência	**Propriedade**	**Localização**	**Acessado por**
Pública	Terceirizada	Provedor	Externa	Usuários externos (não confiáveis)
Privada	Organização	Organização	Local	Usuários internos (confiáveis)
Privada Virtual	Organização e Terceirizada	Provedor	Externa	Usuários internos (confiáveis)
Comunitária	Organização ou Terceirizada	Organização	Local / Externa	Usuários internos
Híbrida	Organização e Terceirizada	Organização Hospedeira	Local / Externa	Usuários internos e externos

A escolha de um determinado modelo depende das necessidades particulares de cada organização, levando-se em consideração questões relacionadas ao custo, segurança e controle. De

um lado, as nuvens públicas oferecem recursos sob demanda mas pouco controle. Do outro lado, nuvens privadas, mais seguras e com maior controle, mas toda a aquisição, gerência, e manutenção é responsabilidade da organização. Entre as duas soluções estão as nuvens híbridas, que podem trazer benefícios dos dois modelos, ou ainda as nuvens comunitárias, voltadas para projetos colaborativos.

2.3 Modelos de serviço

De acordo com o nível de abstração e pelos tipos dos recursos fornecidos pelo provedor, pode-se dividir os modelos de serviço de computação em nuvem em três classes: (1) Infraestrutura como Serviço (*Infrastructure as a Service* - IaaS); (2) Plataforma como Serviço (*Platform as a Service* - PaaS); e (3) Software como Serviço (*Software as a Service* - SaaS). A Figura 2.1, adaptada de Buyya et al. [34], apresenta a organização das camadas que compõem uma nuvem computacional, cada qual fornecendo um conjunto de serviços relacionado a cada modelo.

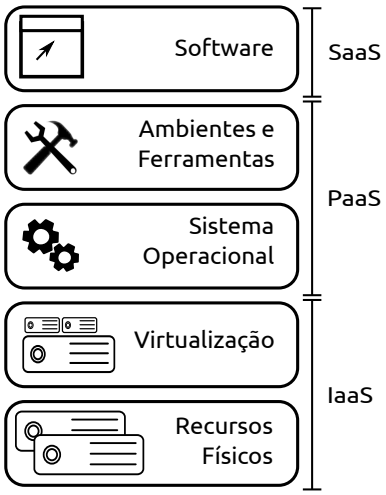

Figura 2.1: Camadas de serviço da nuvem.

Os recursos físicos da nuvem, juntamente com as funcionalidades fornecidas pelos gerenciadores de virtualização, formam a base para entrega de serviço do tipo IaaS. Os recursos são fornecidos por centros de dados (*datacenters*), que contêm um conjunto de recursos físicos interligados, tais como servidores, armazenamento e dispositivos de rede. Estes servidores são transparentemente gerenciados por meio de virtualização, permitindo que suas capacidades sejam compartilhadas entre as instâncias virtuais [34].

A Figura 2.2 mostra o controle exercido pelo provedor e pelo usuário no modelo IaaS. O usuário recebe os recursos requisitados abstraídos por meio de máquinas virtuais, que são geralmente administradas através de protocolos de acesso remoto (SSH, por exemplo). Dessa forma, o usuário não administra diretamente a infraestrutura física da nuvem, mas tem controle total sobre seu ambiente virtual, sendo o responsável por operar, atualizar e configurar os recursos com objetivo de atingir os níveis de desempenho, de segurança e de confiabilidade desejados [101]. Por sua vez, cabe ao provedor manter todos os recursos físicos da nuvem, garantindo seu correto funcionamento e com desempenho satisfatório.

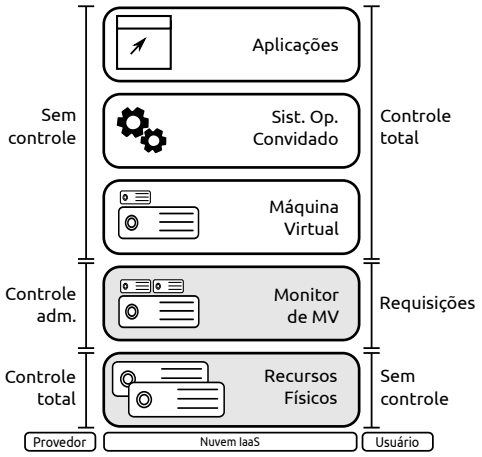

Figura 2.2: IaaS: Escopo do controle.

Atualmente existe um grande número de provedores de IaaS. Ainda que muito semelhantes entre si em relação aos modelos de cobrança adotados, os serviços ofertados e alguns outros pontos podem apresentar pequenas diferenças. Entre os principais provedores de nuvem públicas de IaaS pode-se citar o Amazon [1], Rackspace [15], GoGrid [8] e CloudSigma [4]. Para a implementação de IaaS em nuvens privadas estão disponíveis diversas ferramentas de código aberto, entre elas o Eucalyptus, Nimbus, OpenNebula e OpenStack [123, 128].

A camada intermediária da pilha está relacionada aos serviços de plataforma (PaaS). Nela são fornecidos ambientes de programação, linguagem de alto nível e um conjunto de APIs bem definidas para facilitar a interação entre os ambientes e as aplicações na nuvem [154]. Uma segunda classe de serviços permite a hospedagem e a oferta da aplicação desenvolvida pelo usuário para seus clientes e também oferece as ferramentas para administrar e monitorar as aplicações por eles instaladas.

Como pode ser observado na Figura 2.3, os usuários de PaaS não gerenciam ou controlam a infraestrutura da nuvem. Geralmente possuem acesso apenas às ferramentas de desenvolvimento, a suas aplicações e respectivas configurações.

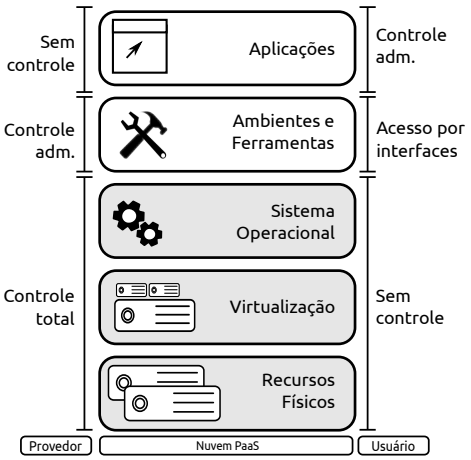

Figura 2.3: PaaS: Escopo do controle.

Comparado com o desenvolvimento de aplicações convencionais, esta estratégia pode reduzir significativamente o tempo de desenvolvimento, oferecendo diversas ferramentas e serviços para uso em um ambiente bem definido. Outra vantagem oferecida pelo modelo de PaaS é que, ainda que os dados estejam fisicamente espalhados pela rede do provedor ou entre vários provedores, toda gerência de dados, incluindo os de desenvolvimento, é vista pelo usuário de forma centralizada. Exemplos de fornecedores de PaaS são o Google Apps Engine [10], o Microsoft Azure [11] e o Salesforce [18]. Como principal desvantagem pode-se citar o fato do usuário ficar dependente da plataforma, não podendo migrar suas aplicações para outra nuvem (*lock-in*).

A camada do topo da pilha oferece aplicações ao usuário final no modelo SaaS. Tais aplicações podem ser acessadas por vários dispositivos (PCs, smartphones, tablets) através de uma interface simples de cliente, tal como um navegador web.

No modelo de SaaS, o cliente não gerencia ou controla a infraestrutura da nuvem, nem mesmo a camada de plataforma, como mostra a Figura 2.4. É possível que em alguns casos o usuário configure parâmetros específicos de personalização. No entanto, o provedor mantém controle administrativo sobre a aplicação, cabendo-lhe a instalação, configuração, atualização e gerenciamento de modo a assegurar o funcionamento adequado da mesma [154].

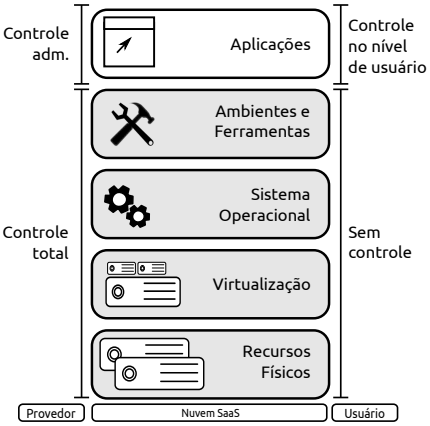

Figura 2.4: SaaS: Escopo do controle.

Este modelo é atraente do ponto de vista do usuário, já que elimina questões de manutenção de software e os custos de suporte e licenças. Além disso, todo o trabalho computacional é executado pela nuvem, não adicionando carga à infraestrutura local do usuário [154].

Os exemplos mais comuns de implementações de SaaS são as aplicações desenvolvidas pela Google (gmail, Docs e Maps) [9], as redes sociais Facebook [6] e Twitter [20] e portais de fotos e vídeos como Flickr [7] e Youtube [23].

A Tabela 2.3 apresenta um resumo dos três modelos de serviço que podem ser oferecidos por uma nuvem, apresentando suas características gerais, vantagens e desvantagens.

Tabela 2.3: Comparação dos modelos de serviço de nuvem

	SaaS	**PaaS**	**IaaS**
Características	Usuários têm acesso a aplicações a qualquer hora e local	Usuários têm acesso a uma plataforma para o desenvolvimento de aplicações que são hospedadas na nuvem	Usuários têm acesso à hardware virtualizado e armazenamento com o qual podem construir sua infraestrutura
Recursos	Aplicações Web e serviços	Ambientes para o desenvolvimento, teste e execução de aplicações	Máquinas virtuais, armazenamento e rede
Controle (Usuário)	Baixo–Nenhum	Médio–Baixo	Alto
Vantagens	Não há a necessidade de instalação e manutenção. Acesso por diferentes dispositivos	Desenvolvimento rápido e padronizado. Foco no desenvolvimento do software. Gerenciamento centralizado	Recursos sob demanda. Manutenção e gerenciamento por conta do provedor (nuvem pública)
Desvantagens	Aplicações pouco flexíveis	Tecnologias restritas *Lock-in*	Preocupações quanto ao isolamento e segurança do ambiente virtual. Gerenciamento semelhante aos centros de dados convencionais
Exemplo de provedores	SalesForce.com (CRM), Clarizen.com (Gerenciamento de projetos), Google Docs (Escritório)	Google AppEngine, Microsoft Azure, SalesForce.com	Amazon EC2 e S3, Rackspace, Joyent, Eucalyptus e OpenNebula

É importante salientar que um provedor não necessariamente precisa ser o proprietário de todas as camadas apresentadas. Por exemplo, um provedor de IaaS pode manter apenas as duas camadas inferiores, enquanto um provedor de SaaS pode conter todas as camadas ou terceirizar

os serviços de armazenamento e processamento. Essa flexibilidade é o que torna a computação em nuvem interessante e tem chamado a atenção nas mais diversas áreas, tanto corporativa, quanto acadêmica.

2.4 Considerações finais

O objetivo deste capítulo foi apresentar uma visão geral sobre a computação em nuvem, de modo a fornecer os conceitos necessários para o entendimento do restante do texto desta tese. De modo geral, o modelo de computação em nuvem pode ser considerado uma combinação e uma evolução de diferentes ideias e conceitos como a computação em grade, computação utilitária e a virtualização.

A computação em nuvem se destaca pela ampla disponibilidade de recursos e pela alocação elástica e sob demanda, que juntas criam a ilusão de fornecer capacidade infinita. Além disso, as nuvens podem fornecer diferentes abstrações ao usuário, oferecendo desde máquinas virtuais (em um nível mais baixo de abstração), até aplicações completas disponibilizadas ao usuário final (em seu nível mais alto), atendendo às necessidades dos mais diversos tipos de clientes e organizações.

Esta flexibilidade na aquisição dos mais diversos tipos de recursos vem chamando a atenção do meio científico e acadêmico para as nuvens computacionais. Já é possível encontrar na literatura técnica diversos casos de adoção do modelo de computação em nuvem, nas mais diversas áreas e finalidades. O Capítulo 3 descreve em detalhes o uso de nuvens no contexto científico, bem como seus benefícios e limitações.

CAPÍTULO 3

COMPUTAÇÃO CIENTÍFICA EM NUVENS

Recentemente, a computação em nuvem tem se mostrado uma alternativa para a execução de aplicações científicas, tanto que muitos pesquisadores têm adotado este novo paradigma para a execução de seus experimentos *in silico*, movendo seus dados e aplicações para a nuvem [59, 73, 138]. O uso de nuvens como ambiente computacional pode ser atraente para a comunidade científica em muitos aspectos, beneficiando desde usuários que possuem pequenas aplicações, até aqueles que executam seus experimentos em centros de supercomputação [131, 108, 120].

Com o intuito de apresentar uma visão geral do uso da computação em nuvens no meio científico, o restante do capítulo está organizado como segue. Na Seção 3.1 apresentam-se os principais paradigmas computacionais utilizados na solução de problemas científicos. O uso das nuvens em ambientes científicos e seus benefícios são apresentados na Seção 3.2. Em contraponto, a Seção 3.3 apresenta alguns problemas e desafios no uso de nuvens para computação científica.

3.1 Computação científica: visão geral

O uso da computação em atividades científicas das mais diversas áreas é cada vez mais comum, de forma que é considerado por alguns autores como o terceiro pilar das ciências, juntamente com a teoria e a experimentação [113]. A Computação Científica é a chave para resolver "grandes desafios" e tem proporcionado avanços e novos conhecimentos nos mais diversos domínios das ciências. Como resultado direto do uso de computadores, muitas áreas de pesquisa têm apresentado progressos mais rápido do que nunca com a realização de experimentos que seriam impossíveis de se realizar há apenas 10 anos.

Uma aplicação científica, de maneira geral, caracteriza-se por se um conjunto diverso de aplicações relacionadas ao processamento de grande volume de dados ou necessidade intensa de processamento. O acesso a recursos computacionais mais rápidos e poderosos possibilita a

resolução de problemas maiores, mais complexos e de modo mais preciso.

Atualmente o uso de arquiteturas paralelas é a solução *de facto* para a execução de aplicações científicas com alta demanda computacionail. Esses sistemas são compostos por máquinas de grande porte ou formados pela agregação de capacidade de processamento, memória, recursos de rede e armazenamento de diversas máquinas independentes e são utilizados de forma coordenada para a execução de aplicações paralelas e distribuídas. Exemplos de arquiteturas são os *clusters* de computadores, as grades e as nuvens computacionais.

Além das diferentes arquiteturas, diferentes paradigmas computacionais podem ser utilizados na construção das aplicações científicas paralelas, dentre os quais pode-se citar a computação de alto desempenho (*high performance computing* - HPC), computação intensiva (*high throughput computing* - HTC) e computação de várias tarefas (*many task computing* - MTC) [110, 81].

A HPC [56] consiste no uso intensivo de recursos de computação por curtos períodos de tempo na solução de problemas de grande porte. Uma aplicação de HPC consiste em um conjunto de tarefas paralelas fortemente acopladas e sincronizadas, geralmente otimizadas para a maximização da eficiência computacional em ambientes dedicados homogêneos e com redes de baixa latência. Para a construção deste tipo de aplicação, é comum o uso de mecanismos de trocas de mensagens (MPI) e o uso extensivo de *multithreading* (Pthreads, OpenMP e CUDA).

Nas aplicações HTC [95] a eficiência computacional é definida em termos da vazão (ou trabalho) ao longo do tempo, ou seja, sem a imposição de prazos rigorosos (podem durar semanas ou meses). Neste tipo de aplicação, divide-se os problemas em um grande número de pequenos subproblemas fracamente acoplados e sem interdependências e, dessa forma, podem ser processados em paralelo sem a necessidade de sincronizações. O Condor [5] é um exemplo de plataforma utilizada na submissão em paralelo de tarefas HTC em recursos distribuídos (*clusters* ou grades computacionais, por exemplo).

Por sua vez, o paradigma MTC é um híbrido entre HPC e HTC [114]. MTC lembra a HTC, mas difere na ênfase do uso massivo de recursos de computação durante curtos períodos de tempo com o objetivo de realizar muitas tarefas computacionais, ou seja, uso de alto desempenho na execução de uma grande quantidade de tarefas fracamente acopladas. A execução de

aplicações MTC são normalmente realizados em recursos computacionais dedicados utilizando *frameworks* como o Falkon [115] e o Kestrel [135].

De modo geral, as nuvens computacionais podem ser utilizadas na execução de aplicações desenvolvidas nos três diferentes paradigmas. No entanto, considerando que muitas das infraestruturas de nuvem atuais não são projetadas especificamente para aplicações científicas, o desempenho pode ser inferior ao obtido em ambientes dedicados, tais como clusters e supercomputadores. Esta questão de desempenho está ligada principalmente ao fato de poucos provedores de nuvem atuais utilizarem redes de baixa latência [72], afetando principalmente as aplicações fortemente acopladas que exigem uma quantidade maior de comunicação e sincronização.

Considerando as características de cada modelo, as aplicações do tipo HTC e MTC são mais apropriadas para os ambientes de nuvem, embora aplicações HPC também possam tirar proveito de outros benefícios oferecidos pelas nuvens, como apresenta a próxima seção.

3.2 Ciência nas nuvens: possíveis usos e benefícios

Nuvens computacionais podem fornecer uma grande variedade de recursos (de hardware e software) que podem ser alocados rapidamente, sob demanda e com custos relativamente baixos, sem a necessidade de aquisição de infraestrutura adicional [142]. Para Oliveira et al. [108], a principal vantagem da computação em nuvem é que o usuário comum é capaz de acessar uma grande variedade de recursos sem ter de adquirir ou configurar uma infraestrutura inteira. Esta é uma necessidade fundamental para aplicações científicas, uma vez que os cientistas não precisam se preocupar com a complexidade do ambiente, concentrando-se apenas em suas atividades de pesquisa.

Atualmente, a lista de experimentos científicos que têm utilizado recursos de nuvem é bastante extensa, mas pode-se citar exemplos nas mais diversas áreas, tais como, no estudo de genomas [96, 152], modelagem atmosférica [59], astronomia [78, 52], processamento de imagens médicas [60, 142], química [77], álgebra linear [105], entre outras. Isso mostra que as nuvens podem ser uma boa alternativa para a demanda de recursos existente no meio científico, principalmente para grupos de pesquisa cujas demandas são temporárias ou que possuem

poucos recursos financeiros para a aquisição de equipamentos [109, 137].

Alguns autores abordam o uso de recursos da nuvem para a complementação de recursos já existentes na organização. No trabalho de Marshall et al. [99] utilizam-se instâncias da Amazon EC2 para estender os recursos de *clusters* virtuais locais criados pelo middleware Nimbus. Bicer et al. [30] descrevem um *framework* para o armazenamento e o processamento de dados usando recursos locais e da nuvem. Calatrava et al. [38] apresentam um estudo sobre o uso de uma infraestrutura híbrida composta por recursos de nuvens e grades para a execução de aplicações científicas. Uma solução semelhante é proposta por Calheiros et al. [39], a qual integra recursos de nuvem e de grades (*Desktop Grids*) para a execução de aplicações da plataforma Aneka.

As nuvens computacionais também têm sido usadas para a redução do tempo de resposta de aplicações científicas [131, 73]. Comumente, a submissão de uma tarefa em um ambiente de alto desempenho é feita em uma fila, cujo tempo de espera pode variar de minutos até dias, dependendo do período que será alocado para a execução do experimento e da disponibilidade dos recursos solicitados. Mesmo não apresentando o mesmo desempenho que em arquiteturas de alto desempenho, algumas tarefas executadas na nuvem podem apresentar tempo de resposta menor quando considerados os tempos de espera da fila [117, 92].

Um outro uso para as nuvens computacionais é descrito por Edlung et al. [57]. Neste trabalho utilizam-se nuvens para a execução de tarefas que não requerem recursos de alto desempenho, mas que acabam sendo alocadas nas filas dos supercomputadores. Segundo o estudo apresentado, cerca de 20% das aplicações que atualmente utilizam os centros de supercomputação nórdicos (distribuídos entre Suécia, Finlândia, Noruega, Dinamarca e Islândia) podem ser enviados para execução em nuvem, reduzindo as filas e reduzindo o tempo de espera das aplicações. Acredita-se que o mesmo ocorra em outros centros de supercomputação espalhados pelo mundo.

A possibilidade de personalização do hardware e do software dos ambientes de execução também é um recurso interessante para ser utilizado no âmbito científico [100]. É comum que aplicações científicas possuam fortes dependências relacionadas com versões de sistema operacionais, ferramentas e bibliotecas, necessitando de compatibilidade entre os ambientes de desenvolvimento, testes e execução efetiva. Devido a essa variabilidade do software suportado

em cada local, configurar e reproduzir o ambiente de execução exato, quando possível, pode requerer muitas horas de trabalho [117]. Com o uso mecanismos de virtualização presentes nos ambientes de nuvem, a pilha de software, o sistema operacional, as bibliotecas, as aplicações e outros dados do ambiente virtual podem ser encapsulados em uma única imagem de máquina virtual. Estas imagens podem ser facilmente transferidas para outras máquinas físicas, salvas para uso posterior ou copiadas para permitir a execução de réplicas de um ambiente [145, 85].

Alguns grupos de pesquisa, como é o caso do CERN [3], estão investigando o uso de imagens de máquinas virtuais para a distribuição de todo o software necessário para a execução de aplicações, exigindo o mínimo esforço para sua configuração. Portanto, basta inicializar a máquina virtual para que um ambiente completo e funcional exigido pela aplicação esteja pronto para utilização, podendo ser replicado a qualquer momento por qualquer outro usuário [117].

Uma outra característica que pode ser explorada pelas aplicações científicas, e que não está disponível em arquiteturas tradicionais, é a elasticidade. De modo geral, as aplicações podem fazer uso da virtualização e da grande disponibilidade de recursos oferecidos pela nuvem para solicitar novos recursos de modo dinâmico e sob demanda. Essa característica é importante para os casos onde os requisitos de hardware necessários não podem ser determinados de modo exato no início da execução ou quando os requisitos alteram-se durante a execução [83]. Mais detalhes sobre elasticidade e exemplos de aplicações elásticas são apresentados no Capítulo 4. A exploração da elasticidade em aplicações científicas é parte do objetivo deste trabalho e é abordada em detalhes no Capítulo 5.

3.3 Ciência nas nuvens: problemas e desafios

Embora as nuvens apresentem diversos benefícios, como discutido até o momento, há algumas questões ainda sem consenso e que impedem a ampla adoção de nuvens computacionais no meio científico, dentre as quais pode-se citar o desempenho, o custo e a segurança.

Diversos autores analisam o desempenho de aplicações científicas em nuvens públicas. Bientinesi et al. [31] apresentam um estudo sobre o desempenho de aplicações matemáticas no Amazon EC2 (*large instances*) no qual constatam que os nodos de processamento, se considerados individualmente, são tão bons quanto os encontrados em sistemas de alto desempenho.

No entanto, as altas latências da rede de interconexão e o compartilhamento de máquinas físicas, que afetam as caches dos processadores, influenciam no desempenho geral da plataforma. Conclusões semelhantes também são apontadas por Evangelinos e Hill [59], He et al. [73] e Rehr et al. [120].

No trabalho de Gupta et al. [72], os autores também analisam a questão do desempenho da rede e concluem que nuvens públicas são apropriadas para aplicações que possuem pouca comunicação, tais como as aplicações HTC e MTC. Para as tarefas fortemente acopladas, como as aplicações desenvolvidas com trocas de mensagens (MPI, por exemplo), ambientes dedicados de alto desempenho ainda são os mais apropriados.

Por sua vez, Church e Goscinski [47] observam o desempenho de nuvens públicas que já dispõem de recursos de alto desempenho e redes de alta velocidade e baixa latência entre elas a nuvem pública Amazon (*Cluster instances*) e duas nuvens privadas com tecnologia VMWare vSphere [21]. Os resultados obtidos pelos autores mostram que o desempenho destas nuvens é similar ao alcançado por um *cluster* com rede InfiniBand. De fato, diversas opções de nuvens de alto desempenho têm surgido recentemente com a proposta de fornecer recursos de processamento, armazenamento e rede apropriados para computações científicas. Alguns exemplos são as nuvens científicas Nebula [12] Helix-Nebula [74] e FutureGrid [64], e as soluções comerciais de alto desempenho oferecidas pela Sabalcore [17] e Amazon [1].

O desempenho de nuvens privadas também é analisado [72, 132, 145]. Por meio de experimentos, constatou-se que o desempenho nestas nuvens é determinado pela infraestrutura subjacente. Isso significa que, mesmo com a presença da camada de virtualização, o desempenho das aplicações é muito próximo se comparado à sua execução no hardware físico disponível.

Uma outra questão relacionada ao uso das nuvens públicas é a variabilidade encontrada no desempenho das máquinas virtuais. Schad et al. [124] relatam que existe uma grande variação no desempenho nas instâncias do Amazon EC2, causada principalmente pela variedade de hardware físico disponível e pela carga em diferentes horários do dia. Essa variabilidade pode fazer com que os tempos de execução de uma aplicação variem significativamente, mesmo alocadas em instâncias de máquina virtual de mesmo tipo. Bientinesi et al. [31] também mostram resultados similares em seus experimentos.

A variação do desempenho em nuvens privadas é abordada nos trabalhos de Rego et al. [119] e Galante et al. [70], nos quais avalia-se a influência a diversidade de modelos de máquinas físicas presentes em uma nuvem privada no desempenho das aplicações. Os autores propõem uma ferramenta para homogenizar o desempenho das máquinas virtuais de nuvens privadas, independentemente do hardware físico utilizado.

A questão da segurança também é abordada na literatura. De acordo com Xiaotao et al. [153], no contexto da computação científica muitos dados são altamente sensíveis, envolvendo propriedade intelectual, informações de planejamento competitivo ou até mesmo questões de segurança e soberania nacional. Nestes casos, segundo o autor, ainda não se pode confiar nos mecanismos de segurança implementados nas nuvens, não importando quão bem intencionados estes possam ser. Portanto, organizações que possuem dados desta natureza devem evitar empregar nuvens para armazenar seus dados e aplicações, sendo os sistemas dedicados mais adequados para estas situações.

O custo também é uma das características importantes quando trata-se de domínios científicos. Como a maioria das nuvens públicas adota o modelo de pagamento pelo uso de sua infraestrutura, é importante estimar o custo final a ser pago e para determinar como os recursos financeiros disponíveis para um experimento científico serão usados [108].

Em geral, o preço a ser pago para a utilização de nuvens pode ser classificado em três tipos principais (que devem ser analisados pelos usuários): gratuito (normalmente, se os cientistas têm a sua própria nuvem), *pay-per-use* (paga um determinado valor relacionado à utilização de recursos da nuvem, normalmente por hora) e por componente (onde paga-se para usar cada tipo de componente, independente do tempo utilizado). No entanto, esta avaliação está longe de ser simples, já que os custos que são reduzidos pela adoção do modelo de nuvem, tais como aquisição de equipamentos e contratação de pessoal de apoio, são difíceis de calcular. Dessa forma, cada situação deve ser analisada individualmente para se verificar qual é a solução mais viável. Considerações a respeito do custo do uso de nuvens para fins científicos são apresentadas por Deelman et al. [52], Vecchiola et al. [142] e Kondo et al. [88], Li et al. [91] e Fox e Gannon [63].

3.4 Considerações finais

De acordo com o apresentado neste capítulo, pode-se concluir que as nuvens atuais ainda não substituem completamente os ambientes dedicados de alto desempenho, mas podem ser vistas como uma alternativa interessante para o fornecimento de recursos de computação necessários para a execução de experimentos científicos, principalmente levando em consideração o acesso bastante restrito de pequenos e médios grupos de pesquisa a computadores de grande porte e redes de alta velocidade.

Pode-se ainda considerar que nestes grupos as necessidades por grandes capacidades computacionais podem ocorrer de modo esparso e por curtos períodos, não compensando o investimento em equipamentos. Dessa forma, o acesso sob demanda aos recursos e a possibilidade de pagamento pelo uso oferecidos pelo modelo de computação em nuvem parece bastante adequado para este cenário.

Uma outra característica oferecida pela computação em nuvem que pode ser utilizada em aplicações científicas é a elasticidade. A elasticidade permite que recursos possam ser adicionados ou removidos do ambiente virtual de modo dinâmico, possibilitando que os recursos sejam ajustados de acordo com a disponibilidade ou demanda. Essa funcionalidade é de grande valia para aplicações científicas que possuem fases distintas ou mudanças em sua estrutura ao longo de sua execução, bem como nos casos onde é difícil saber de antemão quais recursos são os mais apropriados.

O Capítulo 4 descreve o estado-da-arte sobre a elasticidade, apresentando e classificando os principais mecanismos de elasticidade, bem como apontando algumas questões em aberto e possibilidades de pesquisa na área. O Capítulo 5 retoma a questão da elasticidade em aplicações científicas, apresentando as limitações dos mecanismos atuais e propondo uma nova abordagem para a construção de aplicações elásticas.

CAPÍTULO 4

ELASTICIDADE EM NUVENS COMPUTACIONAIS

Nos últimos anos, a computação em nuvem tem atraído a atenção da indústria e do mundo acadêmico, tornando-se cada vez mais comum encontrar na literatura casos de adoção da nuvem por parte das empresas e instituições de pesquisa. Um dos principais motivos é a possibilidade de aquisição de recursos de uma forma dinâmica e elástica. De fato, a elasticidade é um grande diferencial e é atualmente vista como indispensável para o modelo de computação em nuvem [111].

O termo elasticidade é definido como a capacidade de um sistema de adicionar ou remover dinamicamente recursos computacionais utilizados por uma determinada aplicação ou usuário de acordo com a demanda [76]. Os recursos podem incluir desde CPUs virtuais (VCPU[1]), memória, e armazenamento, até máquinas virtuais (MVs) completas. O conceito de elasticidade também pode ser estendido para aplicações. Uma aplicação elástica é aquela capaz de se adaptar às alterações na quantidade de recursos ou de solicitar/liberar recursos de acordo com a sua necessidade. Para que se possa tirar proveito da elasticidade, é necessário que tanto a arquitetura quanto a aplicação sejam capazes de suportá-la de alguma forma.

Muitas vezes o termo elasticidade é usado como sinônimo de escalabilidade, porém são conceitos diferentes e não devem ser usados indistintamente. Escalabilidade é a capacidade de um sistema de ser expandido para atender a futuras necessidades de processamento [133]. Uma aplicação é dita escalável quando a sua eficiência é mantida quando a quantidade de recursos e o tamanho do problema são aumentados proporcionalmente [89]. Comparando as definições é possível identificar as diferenças entre escalabilidade e elasticidade. A escalabilidade é a propriedade que descreve a capacidade do sistema de alcançar uma certa escala (como milhares de servidores ou um milhão de requisições por minuto mantendo sua eficiência). Por outro lado,

[1]Uma VCPU representa um processador virtual atribuído a uma máquina virtual. Similarmente com o que ocorre nos processadores físicos, as VCPUs também podem ser compostas por um conjunto de núcleos (*cores*). No presente trabalho, por questões de simplificação, atribui-se um único núcleo para as VCPUs.

a elasticidade permite que o sistema seja dimensionado sob demanda [24, 50]. Dessa forma, pode haver sistemas ou aplicações escaláveis que não sejam elásticos, sendo o contrário também verdadeiro.

A capacidade de elasticamente expandir e contrair a base de recursos é interessante tanto para o provedor quanto para o usuário final da nuvem. Do ponto de vista do provedor, a elasticidade garante um melhor uso dos recursos de computação, fornecendo economia de escala e permitindo que mais usuários possam ser atendidos simultaneamente, uma vez que os recursos liberados por um usuário podem instantaneamente ser alocados por outro. Da perspectiva do usuário, a elasticidade tem sido utilizada para diversos fins, tais como manutenção da qualidade de serviço [121], complementação de recursos locais [99, 61, 39], redução de custos [129], economia de energia elétrica [130], entre outros.

Recentemente, diversas soluções de elasticidade têm sido desenvolvidas por provedores públicos e pela academia. Na Seção 4.1 apresenta-se uma classificação para as soluções de elasticidade atuais e descrevem-se as principais propostas descritas na literatura técnica com o objetivo de estabelecer o estado-da-arte da elasticidade em nuvens computacionais. Na Seção 4.2 apresenta-se alguns desafios e questões ainda não resolvidas relacionadas à exploração da elasticidade.

4.1 Elasticidade em nuvens computacionais: estado-da-arte

Esta seção tem o objetivo de apresentar o estado-da-arte das soluções de elasticidade oferecidas atualmente. Neste sentido, foram avaliadas 6 infraestruturas de nuvem pública, 3 plataformas para implantação de nuvens IaaS privadas e 28 mecanismos de elasticidade propostos pela academia e pelos provedores de nuvem de IaaS e PaaS.

De modo a sistematizar a apresentação destas soluções, propõe-se uma classificação conforme apresentada na Figura 4.1. Esta classificação foi criada após analisar todas as 33 soluções de elasticidade e extrair suas principais características. Em um primeiro nível, as soluções são separadas em dois grupos: (1) *arquiteturas elásticas* e os (2) *mecanismos de suporte à elasticidade*. No primeiro grupo analisa-se o suporte à elasticidade oferecido pelas infraestruturas de nuvem (IaaS), já no segundo grupo considera-se as características dos mecanismos utilizados

para fornecer elasticidade para as aplicações que são executadas em nuvens IaaS e PaaS.

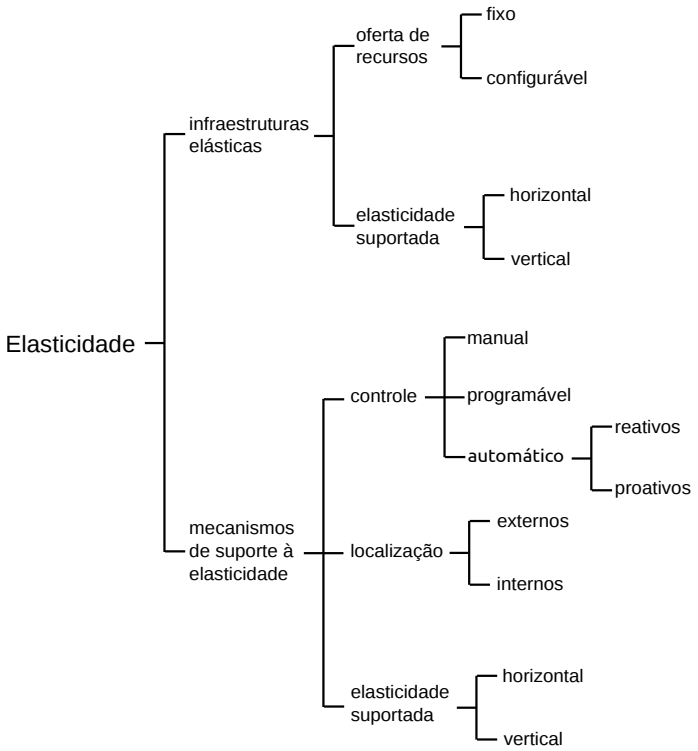

Figura 4.1: Classificação dos mecanismos de elasticidade.

4.1.1 Infraestruturas elásticas

Nos últimos anos, o modelo de nuvem IaaS surgiu como uma alternativa atraente para a aquisição e gerenciamento dos recursos da infraestrutura de computação, permitindo ao usuário requisitar, usar e liberar recursos com uma flexibilidade não encontrada em outros modelos de computação.

De modo geral, as nuvens de IaaS oferecem algum nível de elasticidade que é inerente ao uso de virtualização e da disponibilidade de uma grande quantidade de recursos físicos. No entanto, a maneira como essa elasticidade é oferecida ao usuário final pode variar para cada nuvem de acordo com a *forma como os recursos são ofertados* e qual *tipo de elasticidade é suportada*.

Com relação à oferta de recursos, as nuvens de IaaS podem oferecer recursos de modo *fixo* ou *configurável*. No modo fixo, as MVs são oferecidas como um conjunto pré-definido de CPU, memória e E/S (chamados, por exemplo, de *tipos de instância* pela Amazon e *tamanhos de ser-*

vidor nos provedores GoGrid e Rackspace). A oferta de conjuntos fixos pode ser um problema em casos em que o usuário possui uma necessidade específica que não pode ser mapeada em um dos conjuntos disponibilizados pelo provedor. No modo configurável o usuário pode escolher os recursos de forma personalizada, de acordo com a sua necessidade. Embora seja mais adequado ao conceito de nuvem, o modo configurável está disponível em poucas nuvens de IaaS, como Profitbricks [14] e CloudSigma [4].

Dependendo da forma como a nuvem implementa o provisionamento dos recursos, pode-se classificar a sua elasticidade em horizontal ou vertical [139]. Uma nuvem com elasticidade horizontal (Figura 4.2a.) possibilita apenas a adição ou a remoção dinâmica de MVs da plataforma de computação alocada pelo usuário. Por sua vez, a elasticidade vertical (Figura 4.2b.) é caracterizada pela possibilidade de se alterar a capacidade de MVs em execução. Tipicamente, a elasticidade vertical é implementada através da adição ou remoção de CPUs e memória, mas também pode ser utilizada no contexto de redes e armazenamento. Em alguns casos, a elasticidade vertical também pode ser implementada por meio de migração de MVs. A migração consiste na transferência de uma MV que está sendo executada em um servidor físico para outro servidor físico. Dessa forma, a elasticidade é implementada pela migração de uma MV para uma máquina física que melhor se adequa à carga da aplicação [129], ou ainda, por meio da consolidação e desconsolidação de um conjunto de máquinas em um único servidor hospedeiro [87].

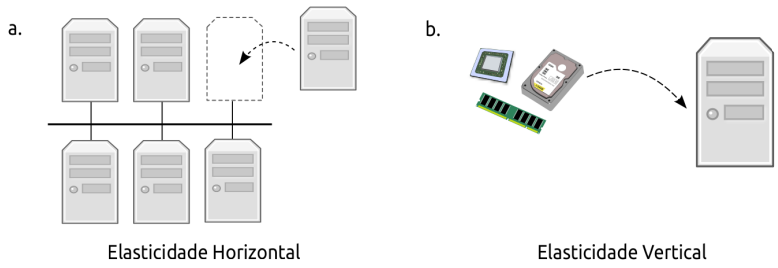

Figura 4.2: Elasticidade horizontal e vertical.

A elasticidade horizontal é o único método suportado pelas nuvens que fornecem modelo de alocação baseado em conjuntos fixos, uma vez que é impossível alterar a configuração de uma MV. Similarmente, o suporte à elasticidade vertical só pode ser oferecido (embora não obrigatoriamente) apenas por aquelas que oferecem recursos configuráveis.

Tanto a elasticidade horizontal quanto vertical deve ser suportada pelo hipervisor utilizado na implementação da nuvem. Atualmente, os principais hipervisores disponíveis oferecem suporte à elasticidade horizontal, permitindo que inúmeras máquinas virtuais possam ser instanciadas. No entanto, o suporte à elasticidade vertical não é igualmente oferecido por todos os hipervisores. A Tabela 4.1 apresenta as características de elasticidade vertical oferecidas pelos principais hipervisores atuais: VMWare vSphere, KVM, Xen e Microsoft HiperV.

Tabela 4.1: Suporte à elasticidade vertical oferecida pelos principais hipervisores.

Hipervisor	**Características**
vSphere	CPU (adição e remoção), memória (via *ballooning*[2]), discos (expansão e adição), interfaces de rede (adição), e PCI express.
KVM	CPU (somente adição), memória (*ballooning*), disco (adição), interfaces de rede (adição).
Xen	CPU (adição e remoção), memória (*ballooning*), discos (expansão e adição), interfaces de rede (adição).
Hiper V	memória (*ballooning*), discos (expansão e adição).

4.1.1.1 Revisão das propostas

De modo geral, as nuvens de infraestrutura atuais, públicas ou privadas, oferecem algum suporte à elasticidade. Esta seção descreve as funcionalidades suportadas por 6 dos principais provedores públicos e 3 plataformas para a construção de nuvens privadas.

A Amazon Web Services [1], um dos maiores e mais tradicionais provedores de nuvem, oferece a seus usuários 17 configurações de máquinas virtuais (tipos de instância), abrangendo desde pequenos servidores de uso geral, até servidores voltados para computação de alto desempenho. Atualmente, suporta plenamente apenas elasticidade horizontal. A Amazon oferece a opção de modificar o tipo de instância, mas a operação exige a reinicialização da máquina. As funções de elasticidade podem ser acessadas por meio de uma API, interface ou por um serviço automático chamado Auto-Scaling (descrito na Seção 4.1.2).

Os provedores Rackspace [15] e Microsoft Azure [11] apresentam funcionalidades similares. Ambos disponibilizam 7 configurações para MVs e suporta apenas elasticidade horizontal. O provedor Rackspace disponibiliza uma API e uma interface para o controle da quantidade de MVs instanciadas, cabendo ao usuário a implementação de mecanismos automatizados mais

elaborados. O Azure, por sua vez, além de oferecer API e interface para o controle da elasticidade lançou no início de 2013 um serviço automático semelhante ao oferecido pela Amazon.

O GoGrid também oferece 7 configurações para MVs, no entanto suporta elasticidade horizontal e vertical apenas para memória. A elasticidade vertical permite que o usuário modifique a quantidade de memória de uma MV utilizando valores pré-determinados. As funções de elasticidade podem ser acessadas por meio de API ou interface.

O ProfitBricks e o CloudSigma são uns dos poucos provedores que permitem ao usuário configurar suas MVs de modo flexível. O ProfitBricks permite ao usuário a criação de MVs com até 62 núcleos de processamento (mapeados diretamente a um núcleo físico) e 240 GB de memória, e suporta elasticidade horizontal e vertical (para CPU e memória) sem a necessidade de reinicialização. O CloudSigma oferece máquinas virtuais com até 80 núcleos (de 1 GHz) e 128 GB de memória, suportando elasticidade horizontal. Ambos fornecem API e interface para o gerenciamento da elasticidade.

As plataformas Eucalyptus, OpenStack e OpenNebula são soluções de software livre para a construção de nuvens privadas e híbridas. Nas duas primeiras, o administrador da nuvem pode criar diferentes configurações de MV, mas estas são fixas para usuário final. Por sua vez, o OpenNebula permite que o usuário configure sua MV com a quantidade de CPUs e memória que desejar. As três plataformas suportam apenas elasticidade horizontal. O Eucalyptus e OpenNebula oferecem interfaces e APIs para o controle da elasticidade. O OpenStack, além dessas opções, também oferece um serviço automático.

A Tabela 4.2 sintetiza as características apresentadas pelos provedores públicos e das plataformas de nuvem privada.

4.1.2 Mecanismos de suporte à elasticidade

Para que se possa tirar o máximo proveito da elasticidade fornecida pela nuvem, não basta que os recursos virtualizados sejam elásticos. Também é necessário que as aplicações tenham capacidade de se adaptar ou serem adaptadas dinamicamente de acordo com alterações em seus recursos. Para tornar isso possível, diversos mecanismos para fornecer suporte à elasticidade para as aplicações têm sido propostos. De acordo com a classificação apresentada na Figura 4.1,

Tabela 4.2: Soluções elásticas e suas classificações.

Nuvem	Oferta de recursos	Elasticidade suportada
Provedores públicos		
Amazon	fixa - 17 configurações	horizontal
Rackspace	fixa - 7 configurações	horizontal
Azure	fixa - 7 configurações	horizontal
GoGrid	fixa - 7 configurações	horizontal e vertical (memória)
ProfitBricks	flexível	horizontal e vertical
CloudSigma	flexível	horizontal
Plataformas para nuvens privadas		
Eucalyptus	fixa	horizontal
OpenStack	fixa	horizontal
OpenNebula	flexível	horizontal

tais mecanismos podem ser classificados de acordo com o *controle*, *implementação* e *elasticidade suportada*.

O *controle* refere-se ao modo de interação necessário para a execução de ações de elasticidade. Se o sistema possui controle *manual*, significa que o usuário é o responsável por monitorar o seu ambiente virtual e suas aplicações, bem como executar todas as ações de elasticidade pertinentes. Neste caso a interação usuário-nuvem é feita com o uso de uma interface.

No controle do tipo *programável*, as ações de elasticidade é feita por meio de chamadas a APIs disponibilizadas pelo provedor da nuvem. Como pode ser visto na Seção 4.1.1.1, praticamente todos os provedores fornecem uma API para alocação e desalocação elástica de recursos. Geralmente estas APIs estão disponíveis para linguagens voltadas à aplicações Web, tais como Java, PHP, Ruby entre outras.

No controle *automático* de elasticidade, o controle e as ações são tomadas por um controlador de elasticidade, de acordo com regras e configurações feitas pelo usuário ou definidas pelo contrato de serviço (SLA), como ilustra a Figura 4.3. O controlador de elasticidade vale-se de informações sobre a carga de trabalho, uso de CPU e memória, tráfego de rede, entre outros, para tomar decisões de quando e quanto escalar os recursos. Estas informações podem ser coletadas por um sistema de monitoramento ou pela própria aplicação.

De acordo com a técnica utilizada pelo sistema de controle para disparar as ações de elasticidade, pode-se subclassificar o controle automático em *reativos* e *proativos*. As soluções *reativas* empregam mecanismos de Regra-Condição-Ação. Nesse caso, as políticas de elasticidade são

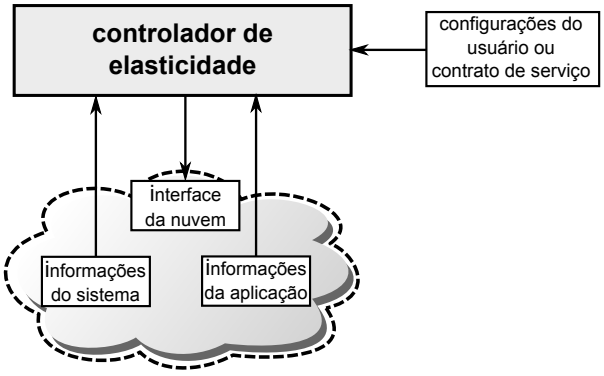

Figura 4.3: Controlador de elasticidade.

definidas em um conjunto de regras e quando uma condição definida em alguma dessas regras é satisfeita, uma ação é disparada [139]. Uma representação desse modelo é apresentada na Figura 4.4.

REGRA:
se CONDIÇÃO(ões) *então* AÇÃO(ões)

CONDIÇÃO:
(1..*) (*se* métrica.valor *igual* limiar) *ou* (*se* evento)

AÇÃO:
(*) ações permitidas pela nuvem (instanciar/remover MV)

Figura 4.4: Mecanismo Regra-Condição-Ação.

O emprego de técnicas reativas é bastante comum e é encontrado em grande parte das soluções propostas pelos provedores públicos e também em vários trabalhos acadêmicos [70].

Por sua vez, as abordagens *proativas* usam heurísticas e técnicas analítico-matemáticas para gerar uma previsão sobre o comportamento das cargas do sistema e, a partir desses resultados, tomar decisões de quando e como escalar os recursos. Segundo Moore et al. [103], a abordagem proativa é mais apropriada para os casos onde a carga de trabalho apresenta padrões bem definidos com periodicidade uniforme, facilitando a previsão de cargas futuras.

A segunda característica analisada é a forma de *implementação* do mecanismo de elasticidade, que pode ser *externa* ou *interna* à aplicação. Os mecanismos externos são implementados como um serviço em separado da aplicação e geralmente utilizam um sistema de monitoramento para coletar informações sobre o ambiente em que a aplicação está sendo executada. Tais informações incluem número de requisições recebidas, uso de CPU e memória, número de clientes conectados, entre outros. Por sua vez, os mecanismos internos são implementados na

própria aplicação, e além das informações do ambiente, também podem disparar ações baseadas em eventos internos à aplicação.

Os mecanismos externos podem ser utilizados em uma determinada classe de aplicações sem a necessidade de alterações, considerando que baseiam-se no monitoramento do ambiente. Já os mecanismos internos são desenvolvidos especificamente para coletar informações particulares de uma aplicação, fornecendo uma solução sob-medida. A implementação de mecanismos internos só é possível se existe uma API de programação que permita que a aplicação tenha acesso à nuvem.

Por fim, analisam-se os mecanismos quanto ao suporte à elasticidade *horizontal* e *vertical*. Se o mecanismo suporta apenas elasticidade horizontal, os recursos são alocados ou desalocados em termos de MVs completas. Neste caso, a aplicação deve ser implementada de modo a assimilar diferentes quantidades de MVs em seu ambiente de execução. Muitas vezes esse suporte é oferecido por mecanismos de balanceamento de carga [139], que tem a função de dividir a carga de trabalho entre as diversas instâncias do servidor, como apresenta a Figura 4.5. Se a carga de trabalho for pequena, apenas um servidor é utilizado (Figura 4.5a.). No entanto, se a carga aumentar, o controlador de elasticidade acrescenta um segundo servidor ao conjunto e o balanceador de carga divide as requisições entre os servidores disponíveis (Figura 4.5b.).

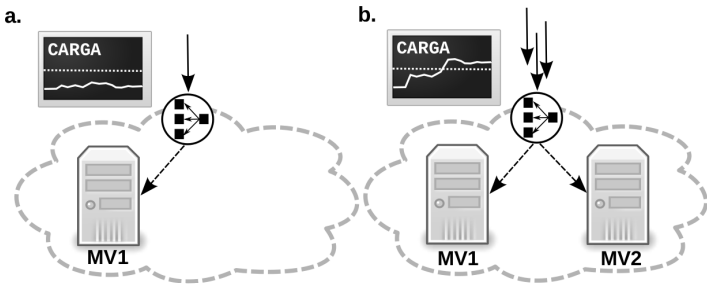

Figura 4.5: Balanceamento de carga.

Se o mecanismo oferece suporte à elasticidade vertical, os recursos são alocados em termos de componentes da máquina virtual, tais como CPU, memória e armazenamento. Alguns tipos de recursos são automaticamente suportados pela aplicação, como no caso da alocação dinâmica de memória ou armazenamento. Por outro lado, a adição de CPUs à MV exige que a aplicação ofereça algum tipo de suporte, tal como uso de múltiplos processos ou *threads*, de modo a utilizar todo o poder de processamento disponível.

4.1.2.1 Revisão das propostas

Nesta seção apresentam-se diversas soluções de elasticidade desenvolvidas para nuvens IaaS e PaaS. De modo geral, a maioria dos provedores públicos de nuvem atuais oferecem alguma funcionalidade de elasticidade, desde os mais básicos até soluções automáticas mais elaboradas. Por sua vez, soluções desenvolvidas no âmbito acadêmico apresentam mecanismos de elasticidade automática bastante semelhantes àquelas utilizadas pelos provedores comerciais, mas adicionando alguma funcionalidade ou empregando metodologias diferentes para o provisionamento elástico de recursos.

A Amazon [1] oferece um mecanismo de elasticidade horizontal – chamado de *Auto-Scaling* – como parte do serviço oferecido por sua nuvem IaaS pública (EC2). A solução é baseada no conceito de Grupo de Auto-Escala (*Auto-Scaling Group* - ASG), que consiste em um conjunto de MVs que podem ser utilizadas para uma determinada aplicação. Para cada ASG existe um conjunto de regras que define a quantidade de instâncias a serem adicionadas ou liberadas. As regras são compostas por gatilhos (*triggers*), que são acionados quando um limiar é alcançado para uma determinada métrica. Os valores das métricas são fornecidos pelo serviço de monitoramento *CloudWatch* e incluem uso de CPU, tráfego de rede, leituras e escritas em disco. Dessa forma, quando o gatilho é disparado, a ação definida pela política correspondente é executada. A solução também inclui balanceadores de carga que são utilizados para distribuir a carga de trabalho entre as instâncias ativas.

O uso de *Spot Instances* oferecidas pela Amazon pode ser visto como uma solução de elasticidade baseada no custo. O serviço funciona como um leilão automático de MVs, no qual os preços flutuam de acordo com a demanda, influenciados, por exemplo, pela hora do dia ou período do ano. O usuário especifica o tipo e a quantidade de instâncias que deseja alocar, e dá um lance com o valor máximo (*bid-value*) que está disposto a pagar por hora de uso de cada instância. Assim, se o lance do usuário for maior ou igual ao valor estipulado pela Amazon, a solicitação será atendida. As instâncias alocadas ficarão disponíveis para o usuário até que este as finalize ou até que o preço da instância supere o lance dado pelas alocações (o que acontecer primeiro).

Os serviços de elasticidade fornecidos pelo Microsoft Azure [11] e pela plataforma de nuvem privada OpenStack [13] utilizam mecanismos automáticos reativos similares ao Amazon Auto-Scaling. Os mecanismos fornecem a capacidade de alocar novas instâncias de MV de forma dinâmica baseada em políticas definidas pelo usuário, indicadores de desempenho e agendamentos, sem qualquer intervenção manual.

Embora grande parte dos provedores e plataformas de nuvem suporte algum tipo de elasticidade, muitos deles não possuem serviços de elasticidade automática. De modo a fornecer essa funcionalidade para tais nuvens, foram desenvolvidas ferramentas como RightScale [16] e Scalr [19].

O RightScale é uma plataforma de gerenciamento que oferece funcionalidades de controle e elasticidade para diferentes provedores de nuvens públicas (Amazon, Rackspace, GoGrid, entre outros) e também para soluções de nuvem privada (CloudStack, Eucalyptus e OpenStack). O RighScale pode utilizar informações do hardware do sistema e também de aplicações, como servidor web Apache e o sistema de banco de dados MySQL.

O Rightscale usa um processo de votação para verificar se adiciona ou remove instâncias do ambiente virtual. Neste processo, a maioria das MVs devem concordar se uma determinada ação deve ser tomada, caso contrário, nenhuma ação ocorre. Cada voto é baseado em um conjunto de regras. Depois de cada ação de escala, há um período de tempo chamado de tempo de redimensionamento (*resize calm time*), onde nenhuma ação pode ser realizada, impedindo a alocação continua de recursos. Recomenda-se 15 minutos para o período de redimensionamento, uma vez que novas máquinas levam entre 5 a 10 minutos para que entrem em funcionamento.

O Scalr é um projeto de código-livre cujo objetivo é oferecer soluções de elasticidade para aplicações web para diversas nuvens, tais como Amazon, Rackspace, Eucalyptus e Cloudstack. Desde 2008 o serviço também está disponível como SaaS com pagamento de assinatura. Atualmente oferece suporte para diversos servidores web e gerenciadores de banco de dados. Seu funcionamento é baseado no monitoramento de métricas de hardware e de aplicações e no disparo de ações baseadas em limiares.

A plataforma proposta por Chapman et al. [44] também implementa um mecanismo reativo. A elasticidade é especificada explicitamente por regras de elasticidade descritas em um formato

baseado no padrão *Open Virtualization Format (OVF)*. As regras especificam condições que consideram eventos de monitoramento da camada de aplicação ou do hardware virtual que são utilizadas pela plataforma para disparar ações específicas. A implementação da plataforma baseia-se em nuvens OpenNebula.

Lim et al. [94] propõem um mecanismo de elasticidade baseado em um intervalo alvo para uma determinada métrica do sistema ao invés de considerar um único limiar para o disparo de ações, como ocorre nas soluções comerciais apresentadas. O objetivo do trabalho é resolver o problema da oscilação na alocação de recursos causados pela existência de um limiar único. A proposta é que o sistema reaja apenas em casos onde a métrica se encontra fora do intervalo previsto, reduzindo a quantidade de alocações e liberações de recurso desnecessárias.

A elasticidade também tem sido usada para estender a capacidade de centros de dados e nuvens privadas. Fitó et al. [61] apresentam o *Cloud Hosting Provider – CHP*, um provedor web elástico que faz uso da terceirização de recursos, tirando proveito das infraestruturas de computação em nuvens públicas para fornecer escalabilidade e recursos de alta disponibilidade para as aplicações implantadas sobre ele. Com o objetivo de evitar violações de SLA, o escalonador do CHP aloca automaticamente recursos em nuvens públicas caso não haja recursos locais suficientes para executar a aplicação.

Marshall et al. [99] descrevem o Elastic Site, uma plataforma para estender clusters usando recursos de nuvens. O sistema foi desenvolvido para utilizar recursos virtuais de uma nuvem privada Nimbus e da Amazon EC2 para fornecer poder computacional adicional aos clusters físicos. Quando os recursos são solicitados ao controlador do cluster (Torque[3]), este pode alocar máquinas físicas do cluster local ou MVs das nuvens, caso não haja recursos físicos disponíveis.

Mecanismos reativos com suporte à elasticidade vertical também têm sido propostos na literatura. O trabalho de Moltó et al. [102] apresenta um mecanismo para elasticidade de memória. O mecanismo monitora o uso de memória da MV e, caso o percentual de memória utilizada esteja fora de determinada faixa, pode alocar ou liberar memória usando a técnica de *ballooning*.

Heo et al. [75] apresenta uma arquitetura para provisionamento elástico de memória e CPU. Nesta plataforma existem dois controladores, um para CPU e outro para memória, que analisam

[3] `http://www.adaptivecomputing.com/products/open-source/torque/`

a taxa de uso de seu respectivo recurso para realizar alocações ou desalocações. A alocação dinâmica de memória utiliza *ballooning* e a alocação de CPU é implementada usando o *Xen Credit Scheduler* [22], com o qual é possível estabelecer a porcentagem de CPU física alocada para cada CPU virtual.

Vijayakumar et al. [143] apresentam um mecanismo de elasticidade para aplicações de fluxo (*streaming*). A proposta consiste na adaptação dos recursos de CPU da MV que hospeda a aplicação de acordo com o fluxo de dados. A aplicação é composta por um *pipeline* de vários estágios, sendo que cada um é alocado individualmente em uma MV. O mecanismo de elasticidade baseia-se na comparação entre a vazão de entrada e a de saída em cada estágio, e se houver um gargalo, aumenta-se a porcentagem de CPU física da MV onde se encontra o problema.

Knauth e Fetzer [87] também abordam aplicações de fluxo de dados. Da mesma forma que apresentado no trabalho anterior, cada estágio do *pipeline* da aplicação é alocado em uma MV, mas ao invés de utilizar o redimensionamento de CPU, emprega-se mecanismos de migração e consolidação para fornecer elasticidade aos recursos. Inicialmente, todas as máquinas são consolidadas em poucos servidores físicos e caso a soma das demandas seja superior à capacidade do servidor hospedeiro, as MVs são migradas para outros servidores, até o ponto em que cada máquina física hospeda uma única MV. Quando a carga for reduzida, todas as MVs voltam a ser consolidadas em um conjunto mínimo de servidores.

Além da abordagem reativa, abordagens proativas têm sido utilizadas para disponibilizar soluções de elasticidade automática. A principal característica dessa classe é o uso de técnicas que preveem o comportamento das cargas do sistema para guiar a tomada de decisões de quando e como alocar os recursos.

Roy et al. [121] utilizam um modelo auto-regressivo de médias móveis (ARMA) para predizer as cargas de trabalho em um sistema. Tal modelo estima a carga com base nos seus valores passados e na média da série de valores, e dessa forma o sistema ajusta a quantidade de MVs necessárias para que o SLA seja garantido.

Vasić et al. [141] descrevem o DejaVu, um *framework* para alocação elástica de recursos de maneira eficiente. A ideia é identificar classes de carga de trabalho e atribuir, para cada uma delas, uma assinatura gerada pela relação entre a carga e os recursos usados para tratá-la

(instâncias da Amazon EC2). Tais assinaturas são geradas em um período de treinamento e armazenadas em uma espécie de cache. Dessa forma, quando uma carga se repete para uma aplicação, a assinatura é buscada na cache e se houver uma entrada compatível, a configuração de recursos relacionada é utilizada.

Sharma et al. [129] apresentam o Kingfisher, um mecanismo preditivo que utiliza-se de mecanismos de replicação de MV e migração de modo a escolher os recursos de melhor custo-benefício. A metodologia empregada consiste em testar empiricamente as diversas opções de servidores físicos e verificar quais delas comportam que tipo de cargas, calculando seus respectivos custos. Usando um método preditivo implementado com o software estatístico R[4], as futuras cargas são estimadas e então, baseado nos custos calculados, decide-se se a MV deverá ser migrada para uma máquina física de maior capacidade ou se novas máquinas virtuais devem ser acrescentadas.

No trabalho de Gong et al. [71], apresenta-se o PRESS, um sistema de elasticidade preditivo baseado em assinaturas geradas com a Transformada Rápida de Fourier (FFT). Para cada padrão de carga de CPU, gera-se uma assinatura que é utilizada na predição do comportamento de aplicações que contém estruturas repetitivas. Para aplicações sem um padrão, emprega-se cadeias de Markov com um número finito de estados para construir uma previsão de curto prazo para os futuros valores de carga. O sistema faz predições a cada minuto e usa as informações geradas para alocar/liberar recursos para a aplicação no próximo minuto. A realocação dos recursos é feita pelo redimensionamento dos limites de CPU pelo *Xen Credit Scheduler*.

O CloudScale [130] é um sistema derivado do PRESS que agrega mecanismos para a redução do consumo de energia. A solução vale-se das características dos processadores modernos de operarem em diversas frequências e voltagens e da capacidade de alterar essas métricas dinamicamente. De acordo com a carga prevista para o sistema, o CloudScale pode reduzir ou aumentar a frequência ou voltagem da CPU, sem desrespeitar os SLO. Por exemplo, se o modelo de predição indicar que a demanda de um determinado servidor físico é 50%, pode-se reduzir a frequência pela metade[5], reduzindo o consumo de energia sem afetar o desempenho

[4] `http://www.r-project.org`

[5] O consumo de um processador é dado por $P = C \times V^2 \times F$, onde P é a energia consumida, C é a capacitância dinâmica dos transistores, V é a voltagem e F é a frequência de operação.

da aplicação.

O uso de técnicas reativas e proativas não são excludentes, como mostra o trabalho de Moore et al. [103], no qual os autores propõem uma abordagem híbrida. Na plataforma proposta, utiliza-se dois controladores, um proativo e um reativo. O controlador proativo utiliza um classificador do tipo Naive Bayes, utilizado para estimar a quantidade de MVs que devem ser alocadas para cada carga de trabalho. O controlador reativo age nos casos onde a previsão é errônea ou quando o controlador proativo deixa de tomar alguma decisão que afeta a qualidade de serviço da aplicação.

Alguns trabalhos utilizam-se de controle manual para disparar ações de elasticidade. Neamtiu [106] apresenta o Elastin, uma plataforma formada por um compilador e um ambiente de execução (*runtime*) cujo objetivo é converter programas originalmente inelásticos em programas elásticos. A ideia por trás do Elastin é o uso de um compilador que permite que diversas versões de uma aplicação inseridas no código-fonte sejam compilados em conjunto, gerando uma única aplicação. Assim, dentro de um único arquivo executável binário podem existir diversas configurações, cada uma apropriada para um determinado cenário. A configuração é definida pelo usuário e pode ser alterada durante a execução caso ocorram mudanças no ambiente de execução.

Rajan et al. [116] apresentam o Work Queue, um *framework* para o desenvolvimento de aplicações mestre-escravo elásticas. Aplicações desenvolvidas com o Work Queue permitem que réplicas de escravos sejam adicionadas em tempo de execução. Os escravos são implementados como arquivos executáveis que podem ser instanciados pelo usuário em máquinas distintas conforme a necessidade. Ao serem executados, os escravos comunicam-se com o mestre que coordena o envio de dados e a execução da tarefa.

No trabalho de Chohan et al. [46] os autores investigam a adição de Spot Instances da Amazon ao longo da execução das aplicações visando a redução de tempo de processamento de aplicações MapReduce. Iordache et al. [80] apresentam uma implementação para MapReduce elástico, chamada Resilin, que suporta alocação dinâmica de recursos de diferentes nuvens, públicas e privadas.

Os mecanismos de elasticidade apresentados até aqui baseiam-se nas decisões dos usuários ou utilizam informações geradas por algum serviço de monitoramento que coleta dados sobre o ambiente virtual para tomar as decisões de quando e como alocar e desalocar recursos (mecanismos externos). Embora sejam menos comuns, pode-se desenvolver mecanismos que levam em consideração informações e eventos gerados internamente pela aplicação (mecanismos internos). Esses mecanismos geralmente são desenvolvidos para uma plataforma ou aplicação específica e exigem que a aplicação tenha a capacidade de se adaptar para utilizar os novos recursos.

Raveendran et al. [118] propõem um *framework* de elasticidade para aplicações MPI. Considerando as limitações das implementações MPI atualmente disponíveis, os autores propõem a adaptação das aplicações por meio do encerramento de uma execução e reiniciando uma nova, usando um número diferente de instâncias virtuais. Vetores e estruturas de dados são então redistribuídos e retoma-se a execução a partir da última iteração. Aplicações que não possuem um laço iterativo não podem ser adaptadas pelo *framework*, uma vez que utiliza-se o índice da iteração como ponto de reinício da execução. A decisão de quando alterar a quantidade de recursos é baseada na expectativa de término da execução. Se a aplicação está evoluindo mais devagar que o esperado, aumenta-se o número de nodos de processamento. Caso evolua mais rápido, pode-se retirar nodos, liberando recursos para outros usuários e diminuir o custo da execução.

Uma outra abordagem para o desenvolvimento de aplicações elásticas é o uso de nuvens de PaaS. Em geral, as aplicações desenvolvidas neste tipo de nuvem possuem elasticidade implícita. Essas nuvens oferecem ambientes de execução (chamados de contêiner), nos quais as aplicações são executadas sem que o usuário se preocupe com os recursos que serão usados. Nesse caso, a nuvem gerencia a alocação dos recursos necessários pelo contêiner de forma transparente. Não há a necessidade de interação do usuário para solicitar mais recursos ou para definições de regras de elasticidade [40].

Um exemplo de plataforma PaaS com suporte à elasticidade é o Aneka [39]. No Aneka, quando uma aplicação necessita de mais recursos, novas instâncias do contêiner são criadas para satisfazer a demanda, usando recursos locais ou recursos de uma nuvem pública, caso os recursos locais sejam insuficientes.

O Google AppEngine [10] é uma plataforma para desenvolvimento de aplicações web escaláveis (Java, Python e JRuby) que rodam no topo da infra-estrutura de servidores do Google. Estas aplicações são executadas dentro de um contêiner e o AppEngine cuida automaticamente da alocação dos recursos necessários. Outras soluções similares são oferecidas pelo Amazon Elastic Beanstalk [1] e pelo servido de PaaS do Microsoft Azure [11].

Finalizando a seção, a Tabela 4.3 apresenta um resumo dos mecanismos de elasticidade apresentados. A ordem dos mecanismos é a mesma que se encontra no texto.

Tabela 4.3: Soluções elásticas e suas respectivas classificações.

Mecanismo	Implementação	Controle	Elasticidade Suportada	Nuvem
IaaS				
1. Amazon EC2 [1]	externa	automático reativo	horizontal	Amazon EC2
2. Amazon Spot-Instances [1]	externa	automático reativo	horizontal	Amazon EC2
3. Microsoft Azure [11]	externa	automático reativo	horizontal	Microsoft Azure
4. RightScale [16]	externa	automático reativo	horizontal	Amazon EC2, Rackspace, GoGrid, Azure, OpenStack
5. Scalr [19]	externa	automático reativo	horizontal	Amazon EC2
6. Chapman et al. [44]	externa	automático reativo	horizontal	OpenNebula
7. Lim et al. [94]	externa	automático reativo	horizontal	*ad hoc*[6]
8. Fitó et al. [61]	externa	automático reativo	horizontal	Emotive
9. Marshall et al. [99]	externa	automático reativo	horizontal	Emotive
10. Moltó et al. [102]	externa	automático reativo	vertical (memória)	OpenNebula
11. Heo et al. [75]	externa	automático reativo	vertical (memória, %CPU)	*ad hoc*
12. Vijayakumar et al. [143]	externa	automático reativo	horizontal	*ad hoc*
13. Knauth e Fetzer [87]	externa	automático reativo	vertical (migração)	*ad hoc*
14. Roy et al. [121]	externa	automático proativo	horizontal	não especificada
15. Vasić et al. [141]	externa	automático proativo	horizontal	Amazon EC2
16. Sharma et al. [129]	externa	automático proativo	horizontal vertical (migração)	OpenNebula
17. Gong et al. [71]	externa	automático proativo	vertical (%CPU)	*ad hoc*
18. Shen et al. [130]	externa	automático proativo	vertical (%CPU, voltagem)	*ad hoc*
19. Moore et al. [103]	externa	automático reativo+proativo	horizontal	não especificada
20. Neamtiu [106]	externa	manual	horizontal	não especificada
21. Rajan et al. [116]	externa	manual	horizontal	*ad hoc*, Amazon EC2 e Azure
22. Chohan et al. [46]	externa	manual	horizontal	Amazon EC2 (Spot-instances)
23. Iordache et al. [80]	externa	manual	horizontal	Amazon EC2
24. Raveendran et al. [118]	interna	programável	horizontal	Amazon EC2
PaaS				
25. Aneka [39]	externa	automático reativo	horizontal (contêiner)	Amazon EC2
26. Google AppEngine [10]	externa	automático reativo	horizontal (contêiner)	Google
27. Amazon Beanstalk [1]	externa	automático reativo	horizontal (contêiner)	Amazon EC2
28. Microsoft Azure (PaaS) [11]	externa	automático reativo	horizontal (contêiner)	Azure

[6]Nuvem construída sem o uso de um gerenciador ou provedor específico.

4.2 Desafios e questões em aberto

Embora muitas soluções de elasticidade tenham sido desenvolvidas por provedores de nuvem e pela academia, algumas questões permanecem em aberto. Nesta seção, apresenta-se alguns dos desafios relacionados ao uso da elasticidade em nuvens computacionais.

A primeira questão está relacionada com a *disponibilidade de recursos*. O compartilhamento dos recursos entre um grande número de assinantes é uma das estratégias utilizadas em nuvens públicas para alcançar baixos custos e economias de escala. Cabe ao provedor manter a qualidade do serviço prestado independentemente da quantidade de usuários que estão se utilizando simultaneamente seus recursos. Dessa forma, muitas vezes os provedores impõem limites rigorosos sobre a quantidade de recursos que um único usuário pode adquirir em cada instante de tempo, negligenciando a premissa dos recursos infinitos [48].

Por exemplo, a Amazon permite que usuários comuns solicitem simultaneamente 20 instâncias sob-demanda e 100 instâncias locais por região. Uma quantidade maior de recursos pode ser alocada, mas depende de uma negociação direta com o provedor. No Rackspace, todas as contas têm um limite pré-configurado de 65 GB de memória total ou cerca de 130 servidores com 512 MB de memória por região.

Atualmente, para a grande maioria dos usuários a cota permitida é maior do que a demanda de suas aplicações (geralmente aplicações web). Conforme aplicações de larga escala começarem a utilizar-se efetivamente de recursos da nuvem, o limite imposto pela disponibilidade de recursos pode afetar a ampla exploração da elasticidade [58]. Para citar um exemplo, é comum que aplicações científicas do tipo *bag-of-tasks* utilizem centenas ou até milhares de nodos.

Uma solução para o problema da disponibilidade de recursos é o uso de múltiplas nuvens. Alguns trabalhos [61, 99] tem abordado a questão do uso combinado de nuvens públicas e privadas, mas o uso combinado de diferentes nuvens públicas ainda é um desafio.

A razão da fraca portabilidade e interoperabilidade entre nuvens é a falta de padrões, uma vez que cada provedor de nuvem possui protocolos e APIs diferentes entre si. Como consequência, a interação e a migração de máquinas virtuais e aplicações entre nuvens distintas é uma tarefa difícil, se não impossível.

Neste sentido, algumas iniciativas estão tentando criar padrões para as plataformas de nuvem. O *Cloud Computing Interoperability Forum* [2], está trabalhando na criação de uma interface de nuvem aberta e padronizada para a unificação de várias API de nuvem. O IEEE [79] também tem um projeto (P2301) de portabilidade e interoperabilidade de nuvem.

Outra perspectiva futura baseia-se na federação de nuvens. Uma federação de nuvens é a implantação e gestão de vários serviços de computação em nuvem internos e externos para atender às necessidades de negócios [41]. Neste cenário, as demandas excedentes de uma nuvem são satisfeitas pelo empréstimo de capacidades computacionais e de armazenamento disponíveis de outros provedores de serviços em nuvem. Algumas arquiteturas para a federação de nuvens foram propostas [35, 144], mas os resultados práticos ainda são preliminares. O desenvolvimento de técnicas e de software para integrar nuvens distribuídas é fundamental para permitir a composição e implantação de serviços de aplicações elásticas.

A segunda questão refere-se à *granularidade dos recursos fornecidos*. Idealmente, os recursos de uma nuvem devem estar disponíveis em qualquer granularidade, permitindo aos usuários alocar dinamicamente desde uma única CPU até um cluster virtual completo, permitindo diferentes níveis de elasticidade [82]. No entanto, a maioria das nuvens IaaS oferece apenas suporte à elasticidade horizontal, na qual os recursos alocados são máquinas virtuais completas. Essas máquinas são oferecidas como um conjunto fixo de CPU, memória e E/S (chamados de *tipos de instância* na Amazon e *tamanhos de servidor* nos provedores GoGrid e Rackspace). Além disso, não há a possibilidade de migração entre os tipos de instância sem a necessidade de reinicialização da mesma.

Assim, a elasticidade fornecida pelos atuais provedores é bastante limitada, e a forma como os recursos são fornecidos pode não refletir a necessidade exata das aplicações, como prevê o modelo de computação em nuvem.

Ben-Yehuda et al. [29] descrevem o cenário perfeito, no qual os recursos de computação, memória e E/S podem ser adquiridos e cobrado por quantidades dinâmicas e não por pacotes fixos. Os clientes deveriam ser capazes de alugar MVs com uma quantidade mínima de recursos e que possam ser continuamente atualizadas apenas com os recursos necessários. Os recursos disponíveis para aquisição incluem processamento, memória e recursos de E/S, bem

como recursos avançados como aceleradores, FPGAs e GPUs. A capacidade de processamento é vendida com base na quantidade de CPUs (ou núcleos) ou por número de ciclos por unidade de tempo, a memória é vendida por megabytes, e dispositivos de E/S são vendidos com base na largura de banda e latência oferecidas.

O terceiro problema está associado ao *tempo de inicialização* e o *tempo de finalização da cobrança* dos ambientes virtuais. A grande vantagem da elasticidade é a capacidade de fornecer recursos dinamicamente em resposta a uma requisição. No entanto, um fato importante neste processo dinâmico é que embora os usuários de nuvem possam fazer seus pedidos a qualquer momento, pode demorar algum tempo para que os recursos adquiridos estejam prontos para uso. Este período de tempo é chamado de inicialização ou *spin-up time* [33].

Em uma nuvem perfeitamente elástica, o provisionamento de recursos é instantâneo, isto é, não existe (ou é insignificante) o atraso entre a solicitação e a alteração da quantidade de recursos. No entanto, nas nuvens do mundo real, o tempo de inicialização pode variar (de 1 a 10 minutos) dependendo de um número de fatores que incluem: o tipo de plataforma de nuvem; tipo de sistema operacional, o número ou o tipo dos recursos solicitados; a disponibilidade dos recursos na região requerida; e a demanda de outros usuários da nuvem. Assim, o provisionamento de recursos pode ser mais lento do que o esperado, afetando a eficácia e eficiência dos mecanismos de elasticidade reais em tratar cargas de trabalho altamente dinâmicas. A Tabela 4.4 mostra o tempo médio de inicialização no Amazon EC2 (m1.small), Azure (Small) e Rackspace (Tipo IV) coletados e apresentados no trabalho de Mao et al. [97].

Tabela 4.4: Tempo médio de inicialização de máquinas virtuais.

Nuvem	**Imagem**	**Tempo médio (seg.)**
EC2	Linux(Fedora) ami-48aa4921	96.9
EC2	Windows (Win Server 2008) ami-fbf93092	810.2
Azure	WebRole default	374.8
Azure	WorkerRole default	406.2
Azure	VMRole - Win Server 2008R2	356.6
Rackspace	Linux (Fedora) flavor 71	44.2
Rackspace	Windows (Win Server 2008R2) flavor 28	429.2

Por sua vez, o *spin-down time* é o intervalo entre o momento que um recurso não é mais utilizado até o momento que não mais se paga por ele [82]. Na Amazon, cada hora parcial

consumida será cobrada como uma hora completa, ou seja, o *spin-down time* pode ser de até 1 hora. No Microsoft Azure, as horas de utilização são faturadas como horas completas para cada hora de relógio. Por exemplo, uma instância inicializada às 10:50 e excluída às 11:10 da manhã, será cobrada por duas horas [11]. Esses modelos de cobrança podem tornar o uso da elasticidade inviável economicamente, principalmente se as demandas ocorrerem em picos ou rajadas de curta duração.

A questão do *spin-up* e *spin-down* será superada com a utilização de novas técnicas de virtualização e com mudanças nas políticas de cobrança dos prestadores de serviços, respectivamente. Alguns trabalhos [156, 136, 51] apresentam técnicas para acelerar o processo de criação de máquinas virtuais, mas até o presente momento, estas técnicas não foram implementadas nos provedores tradicionais. Por sua vez, os problemas relacionados ao *spin-down time* poderia ser resolvido modificando a maneira como os provedores cobram pela utilização de recursos. De acordo com Brebner [33], embora seja improvável que qualquer plataforma de nuvem seja perfeitamente elástica, é possível modelá-lo, assumindo um modelo extremamente refinado de custo no qual se cobra apenas pelos recursos que são realmente consumidos: o byte transmitido, o byte armazenado e o milissegundos do tempo de processamento.

O último problema a ser tratado é a falta de ferramentas e *frameworks* genéricos para a construção de aplicações elásticas. Como apresentado na na Seção 4.1.2, a maioria das soluções de elasticidade tem como alvo aplicações comerciais cliente-servidor (servidores web, banco de dados e servidores de e-mail), enquanto um pequeno conjunto de propostas tem como foco modelos específicos de aplicação (*Streaming*, MPI, *Workflows*, MapReduce e mestre-escravo).

Dessa forma, é possível constatar que ainda faltam ferramentas e mecanismos que forneçam suporte apropriado para o desenvolvimento de aplicações em diversos modelos de programação e para a exploração da elasticidade de modo mais amplo, por exemplo, permitindo a alocação de recursos em diferentes granularidades.

4.3 Elasticidade na Computação Científica

Graças à sua capacidade de fornecer uma ilusão de recursos ilimitados ou imediatamente disponíveis, as nuvens podem possibilitar o desenvolvimento de aplicações científicas com novos

e interessantes modos de execução, que permitem que novos recursos (MVs, CPUs, memória, armazenamento, entre outros) possam ser adicionados ou removidos de acordo com as reais demandas.

Essa característica é de grande valia para aplicações dinâmicas, isto é, aquelas que possuem necessidades de recursos que não podem ser determinadas com exatidão, quer devido a mudanças nos requisitos em tempo de execução ou devido a mudanças na estrutura da aplicação (por exemplo, método de soluções com diferentes demandas)[83].

O fornecimento de máquinas virtuais completas (elasticidade horizontal) beneficiam principalmente as aplicações do tipo HTC e MTC, uma vez que estas são fracamente acopladas e o acréscimo de nodos de processamento permite que novos processos trabalhadores (ou escravos) possam ser instanciados dinamicamente.

Já o uso da elasticidade vertical pode ser interessante para uma variedade ainda maior de aplicações. A alocação dinâmica de CPUs (núcleos) pode ser usada por aplicações que possuem fases distintas de processamento, tais como aplicações *multithread*, que em determinadas etapas de sua execução utilizam apenas um único núcleo de processamento, enquanto que em outras pode utilizar uma quantidade massiva de núcleos.

O uso de alocação dinâmica de memória é abordada por Moltó et al. [102]. Segundo o autor, o provisionamento elástico de memória permite ajustar a memória alocada para a máquina virtual de acordo com a real demanda da aplicação, evitando a ocorrência de *thrashing* e reduzindo o excesso de memória não utilizada.

De modo similar, pode-se explorar o uso de armazenamento elástico por meio da alocação de unidades de armazenamento adicionais conforme a demanda da aplicação, por exemplo, durante a escrita de arquivos de saída onde constatou-se a necessidade de espaço adicional.

De modo geral, o uso de recursos elásticos pode beneficiar aplicações de diferentes formas, dependendo do modelo e da forma como são implementadas. É possível encontrar na literatura alguns exemplos do uso da elasticidade para fornecer algumas funcionalidade interessantes para aplicações científicas com diferentes características (vide Seção 4.1.2.1 e os resultados apresentados nos Capítulos 6 e 7).

4.4 Considerações finais

Esta seção teve como objetivo apresentar os principais conceitos relacionados à elasticidade em nuvens computacionais, bem como levantar o estado-da-arte do assunto. Apresentou-se ainda uma proposta de classificação para os mecanismos de elasticidade, elaborada com base nas características das soluções e trabalhos analisados. Por fim, foram descritos alguns desafios relacionados à exploração da elasticidade em nuvens que podem ser utilizados para guiar futuras pesquisas na área.

De acordo com o apresentado, a elasticidade já é implementada, mesmo que de modo limitado, pela maioria dos provedores de nuvem atuais com o intuito de reduzir os problemas de provisionamento que ocorrem no modelo tradicional de centro de dados. No meio acadêmico, a elasticidade tem sido alvo de estudos com diversas finalidades, dentre as quais pode-se citar o aumento da capacidade de recursos locais, economia de energia e redução de custos de execução de aplicações.

Embora uma grande quantidade de soluções tenham sido propostas, nota-se que estas ainda são limitadas a alguma plataforma ou modelo de programação e não permitem uma ampla exploração da elasticidade oferecida pelo modelo de computação em nuvem. A falta de mecanismos de elasticidade abrangentes, mais especificamente no contexto de aplicações científicas, é o alvo da pesquisa deste trabalho e é abordada com mais detalhes no Capítulo 5.

CAPÍTULO 5

EXPLORANDO A ELASTICIDADE EM NÍVEL DE PROGRAMAÇÃO

Conforme apresentado no Capítulo 4, diversos mecanismos de elasticidade estão disponíveis atualmente. No entanto, estes mecanismos ainda apresentam uma série de limitações em fornecer suporte adequado para aplicações científicas, uma vez que não foram desenvolvidos para este fim. Neste contexto, este capítulo apresenta uma abordagem para a exploração da elasticidade em aplicações científicas, na qual o controle da elasticidade é feito em nível de programação. Tal abordagem permite que as particularidades de cada aplicação possam ser consideradas na construção de controladores de elasticidade sob medida e que novas funcionalidades sejam agregadas às aplicações.

O capítulo está dividido conforme descrito a seguir. Na Seção 5.1, apresentam-se a motivação e as justificativas para o desenvolvimento da abordagem descrita. Na Seção 5.2 descreve-se em detalhes a exploração da elasticidade em nível de programação. Por fim, a Seção 5.3 apresenta o Cloudine, um *framework* desenvolvido com o objetivo de fornecer suporte à construção e à execução de aplicações científicas elásticas empregando a abordagem proposta.

5.1 Motivação e justificativa

Embora as nuvens sejam ambientes muito flexíveis e de fácil acesso, seu uso para atividades científicas não é simples, e sua efetiva utilização por parte dos pesquisadores ainda apresenta desafios. Isso se deve principalmente ao fato de que as propriedades mais atrativas do paradigma de computação em nuvem para o contexto acadêmico, dentre as quais se encontra a elasticidade, ainda não terem sido plenamente implementadas nas soluções atuais [49].

De fato, no contexto da elasticidade, os mecanismos disponíveis atualmente apresentam uma série de limitações ao fornecer elasticidade para aplicações científicas de modo amplo [151, 67]. Conforme apresentado no Capítulo 4, a maioria dos mecanismos foi desenvolvida para escalar aplicações cliente-servidor (servidores web, e-mail e banco de dados) de acordo com a oscilação

de suas cargas de trabalho [45].

O controle da elasticidade nesses mecanismos é realizado por um controlador externo que se utiliza de dados coletados por um sistema de monitoramento que são usados em um conjunto de regras ou na predição do comportamento da aplicação para decidir sobre a adição ou remoção de recursos. Os dados incluem o número de clientes conectados, o número de requisições ao servidor, o uso de CPU e memória das MVs e as operações de entrada e saída. Em geral, a elasticidade oferecida baseia-se na variação do número de máquinas virtuais empregadas pelos componentes da aplicação e no uso de balanceadores de carga para dividir a carga de trabalho entre os diversos servidores virtualizados.

Um exemplo de uso dessa abordagem é apresentado na Figura 5.1, adaptada de Roy et al. [121], na qual é possível observar a alocação de MVs em função dos clientes conectados. O controlador de elasticidade usa o número de clientes para alocar ou liberar dinamicamente MVs de modo a tratar a variação de carga de trabalho.

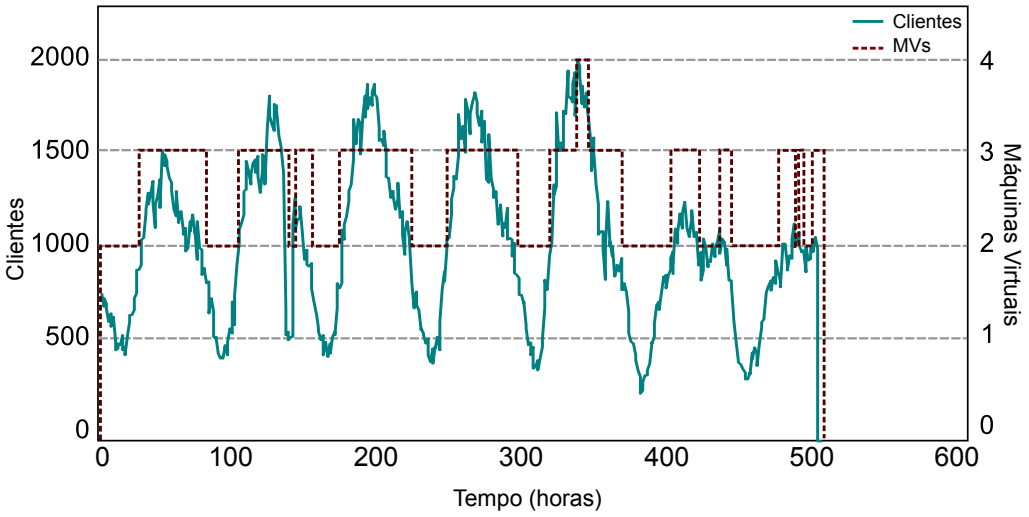

Figura 5.1: Uso da elasticidade em uma aplicação web.

A primeira restrição ao uso deste tipo de mecanismo para fornecer elasticidade para aplicações científicas se deve ao fato de que tais aplicações são geralmente projetadas para usar uma quantidade fixa de recursos que é definida em sua instanciação, não podendo explorar a elasticidade sem o suporte apropriado [118]. Dessa forma, a adição de máquinas virtuais e uso de balanceadores de carga não tem efeito nestas aplicações, uma vez que elas não são aptas a detectar e usar os recursos adicionais.

Outro problema se dá pelas diferenças entre os padrões das cargas de trabalho das aplicações baseadas em servidor e das aplicações científicas. Segundo Armbrust et al. [25], um servidor que hospeda aplicações como web e banco de dados tem carga média que varia entre 5% a 20% com a presença de picos eventuais causados por fatores externos (clientes ou requisições). Essa variação permite que o controlador de elasticidade detecte comportamentos fora do padrão e aloque os recursos necessários.

Por outro lado, aplicações científicas, em sua grande maioria, são executadas em lote (*batch*) e a sua carga de trabalho é determinada pela entrada utilizada (geralmente arquivos) e por um conjunto de parâmetros, inviabilizando o uso de mecanismos baseados no monitoramento de requisições externas, uma vez que essas são inexistentes [151]. Além disso, aplicações científicas tendem a utilizar intensivamente todos os recursos de processamento atribuídos no início de sua execução. Um exemplo é apresentado na Figura 5.2, na qual é possível observar o uso de CPU para uma aplicação numérica utilizando um, dois e quatro *threads*. Note que as CPUs são totalmente utilizadas[1] em praticamente toda a execução da aplicação, independentemente do número de *threads* usados. Assim, o uso de mecanismos baseados na utilização de CPUs também se torna ineficaz, uma vez que podem sinalizar erroneamente a necessidade de novos recursos. O mesmo é valido para memória e armazenamento.

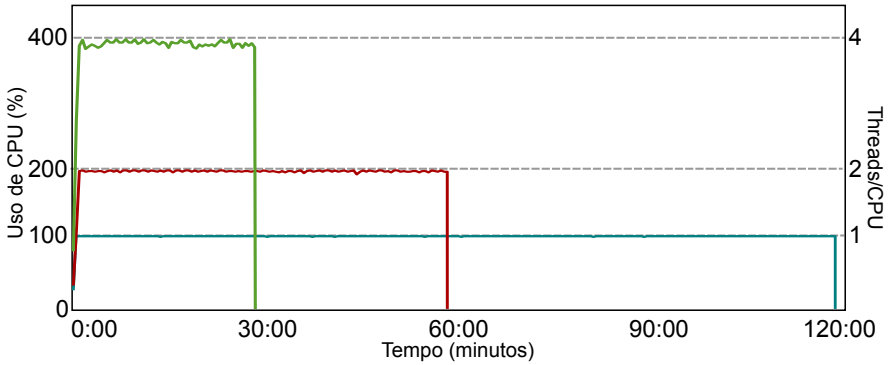

Figura 5.2: Uso de CPU com diferentes quantidades de *threads*.

Deve-se ainda considerar o fato que os sistemas de monitoramento coletam informações de hardware das máquinas virtuais, mas são incapazes de obter dados sobre os eventos gerados internamente pela aplicação e que podem demandar a alocação elástica de recursos, tais como, a criação de um novo processo/*thread* ou a alocação de uma nova estrutura de dados.

[1] Porcentagens acima de 100% indicam que múltiplas CPUs estão sendo utilizadas.

Considerando as restrições apresentadas, alguns trabalhos apresentam mecanismos de elasticidade para aplicações científicas em nuvens IaaS. No entanto, esses mecanismos ainda são limitados a determinados modelos de aplicação, tais como, workflows [37], MapReduce [46, 80], troca de mensagens (MPI) [118] e mestre-escravo [116]. Pode-se citar ainda o trabalho de Moltó et al. [102] que descreve um mecanismo para alocação dinâmica de memória para aplicações científicas baseado no monitoramento da máquina virtual. Experimentos apresentados nos Capítulos 6 e 7 mostram que a abordagem adotada é ineficiente em casos onde grandes quantidades de memória são requisitadas rapidamente.

Outra questão é que os mecanismos atuais implementam apenas elasticidade horizontal. É possível comprovar este fato ao se observar que 78% dos mecanismos listados no Capítulo 4 não oferecem qualquer suporte à elasticidade vertical. Embora a elasticidade vertical seja mais restrita, pois limita-se à capacidade de uma única máquina física, o seu uso é de grande interesse dada a disponibilidade cada vez maior de computadores com múltiplos núcleos de processamento. Considera-se ainda que o redimensionamento dos recursos de uma MV é muito mais rápido que a instanciação de uma MV completa, e que o suporte à elasticidade vertical pode permitir que mais tipos de aplicação tirem proveito da alocação dinâmica de recursos (*multithread*, por exemplo).

Dado este panorama, na próxima seção, propõe-se uma abordagem para exploração da elasticidade em aplicações científicas com o objetivo de superar as limitações expostas nos mecanismos atuais de elasticidade.

5.2 Elasticidade em nível de programação

Considerando o exposto na seção anterior, o uso dos mecanismos atuais de elasticidade em aplicações científicas é limitado por 4 fatores principais: (1) As aplicações não são capazes de detectar e usar recursos elásticos; (2) Os mecanismos atuais não conseguem coletar informações provenientes da aplicação; (3) Em sua maioria, os mecanismos empregam apenas elasticidade horizontal, e; (4) As poucas soluções que tem como alvo aplicações científicas são restritas a um determinado modelo. Para que a elasticidade possa ser amplamente empregada em aplicações científicas é necessário que essas quatro limitações sejam suprimidas.

Neste sentido, propõe-se uma abordagem para a exploração da elasticidade, na qual o controle da elasticidade é feito em nível de programação, o que significa que o controlador de elasticidade é incorporado ao código-fonte, permitindo que as ações de alocação e desalocação de recursos partam da própria aplicação, sem a necessidade de mecanismos externos ou de interação com o usuário.

Conforme ilustrado na Figura 5.3, ao se mover o controlador da elasticidade para dentro da aplicação, este passa a ter acesso a todas as informações internas, enquanto que os mecanismos baseados em monitoramento coletam apenas informações sobre as cargas de trabalho e sobre o estado da máquina virtual. Dessa forma, a lógica do controle da elasticidade pode ser elaborada de modo a considerar os eventos internos, os parâmetros de configuração, as entrada de dados, entre outros. Por exemplo, pode-se optar por adicionar novas VCPUs quando *threads* são criadas ou alocar mais memória para armazenar uma nova estrutura de dados.

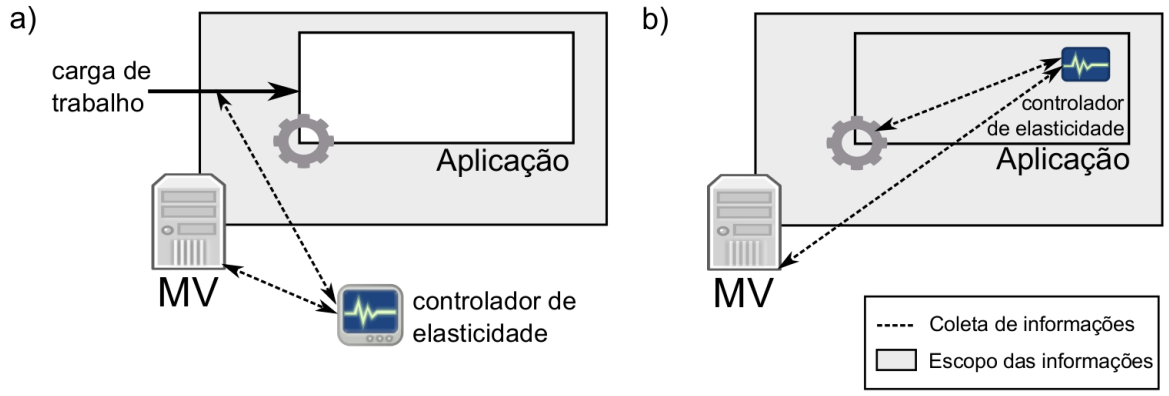

Figura 5.3: Sistema de monitoramento (a) versus controle incorporado na aplicação (b).

Considerando que, na abordagem proposta, a coleta de informações e as ações de elasticidade fazem parte da implementação da aplicação, devem ser oferecidos mecanismos apropriados para tais tarefas. Neste trabalho, propõe-se o uso de *primitivas de elasticidade*, que correspondem a um conjunto de funções básicas que permitem a comunicação com a infraestrutura de nuvem subjacente para a solicitação ou liberação de recursos, bem como coletar informações do ambiente virtual.

A Figura 5.4 ilustra o funcionamento das primitivas na alocação dinâmica de recursos. Quando a primitiva é executada, uma requisição é enviada para a nuvem solicitando os novos recursos. Se há recursos disponíveis, estes são alocados ao ambiente virtual. Neste exemplo, solicita-se a adição de 2 VCPUs, que em seguida são alocadas à máquina virtual em que a aplicação está sendo executada.

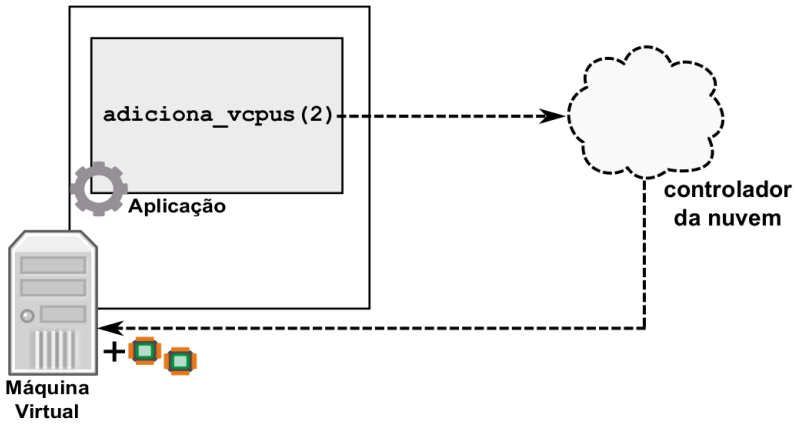

Figura 5.4: Alocação dinâmica de recursos usando primitivas de elasticidade.

O conjunto de primitivas deve ser abrangente o suficiente para permitir a ampla exploração da elasticidade. Isso significa que devem ser oferecidas primitivas para o emprego de elasticidade horizontal, permitindo a alocação e desalocação de máquinas virtuais completas, bem como para o uso de elasticidade vertical, permitindo a reconfiguração das máquinas pela adição ou remoção dos componentes que as compõem, tais como VCPUs, memória e disco. Deve-se contemplar também primitivas para a coleta de informações do ambiente virtual e da infraestrutura da nuvem. Tais informações são indispensáveis para a elaboração dos controladores de elasticidade, uma vez que é por meio destas que determina-se a necessidade de novos recursos e se há disponibilidade para alocação. No caso de nuvens públicas podem também ser oferecidas informações a respeito do custo para a utilização dos recursos.

É importante destacar que nem todas as nuvens suportam todos os tipos de primitivas. Por exemplo, o Amazon EC2 não suporta a alocação dinâmica de memória ou VCPUs, suportando apenas a alocação e desalocação de instâncias de máquinas virtuais. Assim, o conjunto de primitivas que pode ser efetivamente utilizado por uma aplicação depende das características da nuvem subjacente.

A possibilidade de considerar a alocação de recursos como parte da lógica do programa dá origem a um novo paradigma para o projeto e o desenvolvimento de aplicações. Neste paradigma, os recursos passam a ser tratados como elementos variáveis de um programa, podendo ser instanciados e modificados ao longo da execução. Isso permite que o programador desenvolva e integre à aplicação controladores de elasticidade que consideram as características particulares das aplicações, tais como modelo de programação, eventos internos, dados de entrada e seus parâmetros de configuração.

Como consequência, funcionalidades inéditas, não previstas nos mecanismos de elasticidade de uso geral, podem ser agregadas às aplicações científicas. Pode-se desenvolver aplicações dinâmicas e flexíveis que adaptam o seu próprio ambiente de execução de acordo com sua estrutura lógica e demandas, de modo a aprimorar seu desempenho, melhorar o uso dos recursos, reduzir o custo de execução, ou ainda, tirar vantagem de recursos ociosos ou de baixo custo. É possível ainda modificar aplicações legadas, bibliotecas e *frameworks* de programação paralela (originalmente projetados para suportar uma quantidade fixa de recursos ao longo da execução) para que suportem a elasticidade fornecida pelos ambientes de nuvem. Aplicações com estas características são apresentadas nos trabalhos de Galante e Bona [68, 69] e também nos experimentos dos Capítulos 6 e 7.

Revisitando as quatro limitações apresentadas no início desta seção, pode-se afirmar que todas elas são superadas na abordagem proposta. A primeira limitação é abolida uma vez que as aplicações não são só capazes usar recursos adicionais, mas também ganham a capacidade de solicitar ou liberar seus próprios recursos. Elimina-se a segunda limitação ao mover o controlador de elasticidade para dentro do código-fonte da aplicação. Dessa forma, usando as primitivas adequadas, pode-se construir controladores que levam em consideração as demandas e os eventos gerados internamente nas aplicações. A terceira limitação também é superada na abordagem proposta, considerando que o conjunto de primitivas deve oferecer a possibilidade de explorar a elasticidade tanto horizontal quanto vertical. Note, porém, que o uso de ambos os tipos de elasticidade depende do suporte oferecido pela nuvem. Por fim, a exploração da elasticidade em nível de programação permite que aplicações elásticas em diversos modelos possam ser implementadas, já que é possível desenvolver soluções sob medida para cada aplicação.

De modo a sintetizar o apresentado nesta seção, a Tabela 5.1 apresenta um comparativo dos aspectos presentes na abordagem utilizada pelos mecanismos tradicionais e na abordagem proposta neste trabalho. Comparando-se as características, são evidentes as vantagens da exploração da elasticidade em nível de programação no desenvolvimento de aplicações científicas.

Tabela 5.1: Comparação entre abordagens utilizadas na exploração da elasticidade

Característica	**Tradicional**	**Proposta**
Uso em aplicações científicas	Quando possível, geralmente limitado a um único modelo de aplicação	Sim. Projeto voltado para este fim. Pode-se desenvolver aplicações em diversos modelos
Tipos de elasticidade suportados	Grande parte dos mecanismos suporta apenas elasticidade horizontal	Oferece a possibilidade de empregar tanto elasticidade vertical e horizontal, mas depende da nuvem utilizada
Controle da elasticidade	Baseado em informações coletadas a partir de sistemas de monitoramento ou da interação manual do usuário	A decisão sobre alocar ou desalocar faz parte da lógica da aplicação
Conseguem coletar informações da aplicação?	Não. Consideram apenas cargas de trabalho externas e dados da MV (uso de CPU, memória, por exemplo). Uma exceção é o trabalho de Raveendran et al. [118]	Sim, os dados de entrada e parâmetros, bem como eventos internos podem ser considerados
Flexibilidade	Não há. Os mecanismos oferecem soluções prontas ao usuário, o qual pode configurar alguns parâmetros	O programador tem a possibilidade de desenvolver soluções sob medida para cada aplicação. Pode-se desenvolver aplicações elásticas, assim como adaptar bibliotecas e *frameworks* já existentes

Para validar a abordagem proposta, desenvolveu-se o *Cloudine*, um *framework* que possibilita a construção e a execução de aplicações científicas elásticas em nuvens IaaS. O *framework* é apresentado sucintamente na Seção 5.3 e com mais detalhes no Apêndice A.

5.3 Cloudine

O Cloudine (*Cloud Engine*) é o *framework* desenvolvido com os objetivos de oferecer o suporte necessário para o desenvolvimento de aplicações científicas usando controle de elasticidade em nível de programação.

Para possibilitar que as aplicações comuniquem-se com a infraestrutura da nuvem, o *framework* emprega dois componentes principais: o Ambiente de Execução (*runtime environment*) e a API de Elasticidade, conforme ilustra-se na Figura 5.5.

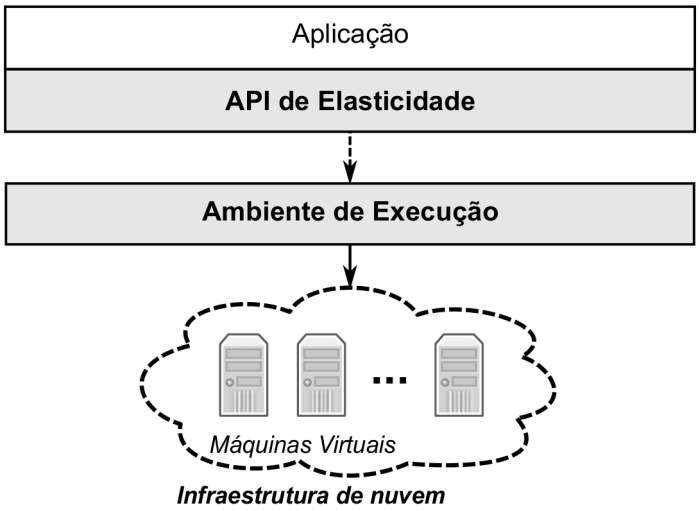

Figura 5.5: Arquitetura do *framework* Cloudine.

O Ambiente de Execução é o componente que gerencia o provisionamento dinâmico de recursos e realiza toda a interação entre as aplicações elásticas (via API) e infraestrutura de nuvem. É por meio deste componente que as solicitações são processadas e repassadas à nuvem subjacente.

Por sua vez, a API de Elasticidade fornece um conjunto de primitivas que permitem a construção de aplicações elásticas. As primitivas implementam a comunicação entre a aplicação e o Ambiente de Execução, que é utilizado para a realização da alocação ou liberação de recursos. Considerando a proposta do *framework*, a API oferece primitivas para a exploração da elasticidade horizontal e vertical, bem como primitivas de coleta de informações.

No Cloudine, uma aplicação elástica é construída a partir da inserção de primitivas de elasticidade, que quando executadas comunicam-se com Ambiente de Execução para enviar uma requisição. Esta requisição é então convertida pelo Ambiente de Execução em um conjunto de comandos para a nuvem.

Para permitir a realização de experimentos práticos, implementou-se um protótipo funcional do *framework*. A implementação do Ambiente de Execução foi feita utilizando-se as linguagens C, Python e Shell Script e emprega o banco de dados SQLite para o armazenamento das informações dos ambientes virtuais. A comunicação entre os componentes foram implementadas usando sockets TCP/IP. Até o presente momento, o protótipo oferece suporte para nuvens OpenNebula com virtualização Xen.

Desenvolveu-se também uma API de elasticidade com suporte às linguagens C e C++. Até o presente momento, a API implementa 12 primitivas para a alocação elástica de VCPUs, memória, e máquinas virtuais completas, bem como para coleta de informações sobre o ambiente virtual e sobre a infraestrutura da nuvem. A API é disponibilizada por meio da biblioteca compartilhada dinâmica (*dynamic shared library*) `libclne.so`. A Tabela 5.2 apresenta as primitivas implementadas e suas descrições.

Tabela 5.2: Primitivas implementadas pela API de Elasticidade

Função	**Descrição**
`int clne_add_vcpu(int N)`	Adiciona *N* VCPUs para a MV na qual a aplicação está sendo executada. Retorna 1 em caso de sucesso, 0 caso contrário.
`int clne_rem_vcpu(int N)`	Remove *N* VCPUs da MV na qual a aplicação está sendo executada. Retorna 1 em caso de sucesso, 0 caso contrário.
`int clne_add_node(int N)`	Adiciona *N* nodos ao ambiente virtual (cluster). Retorna 1 em caso de sucesso, 0 caso contrário. Esta função também cria (ou atualiza) um arquivo contendo os nomes e endereços de IP dos nodos que compõem o cluster.
`int clne_rem_node(int N)`	Remove o nodo atual do ambiente virtual (cluster). Retorna 1 em caso de sucesso, 0 caso contrário.
`int clne_add_memory(long int N)`	Adiciona *N* Megabytes de memória para a MV na qual a aplicação está sendo executada. Retorna 1 em caso de sucesso, 0 caso contrário.
`int clne_rem_memory(long int N)`	Remove *N* Megabytes de memória da MV na qual a aplicação está sendo executada. Retorna 1 em caso de sucesso, 0 caso contrário.
`int clne_get_freemem()`	Retorna a quantidade de memória livre da máquina física que hospeda a MV na qual a aplicação está sendo executada.
`int clne_get_maxmem()`	Retorna a quantidade total de memória da máquina física que hospeda a MV.
`int clne_get_mem()`	Retorna a quantidade de memória livre da MV na qual a aplicação está sendo executada.
`int clne_get_freecpu()`	Retorna a quantidade de CPUs livres na máquina física que hospeda a MV.
`int clne_get_maxcpu()`	Retorna a quantidade total de CPUs da máquina física que hospeda a MV.
`int clne_get_vcpus()`	Retorna a quantidade de CPUs da MV na qual a aplicação está sendo executada.

O Cloudine pode ser utilizado de duas formas na construção de aplicações elásticas. Na primeira, utiliza-se diretamente a API de elasticidade na implementação da aplicação, cabendo ao programador inserir corretamente a lógica de controle de elasticidade. A outra forma consiste em utilizar a API para adaptar bibliotecas e *frameworks* já consolidados de modo a adicionar suporte à elasticidade a elas. Assim, aplicações elásticas podem ser obtidas transparentemente a partir destes *middleware* modificados.

Experimentos envolvendo o uso direto do Cloudine, bem como seus resultados são apresentados no Capítulo 6. Para demonstrar o uso do *framework* na adaptação de bibliotecas de programação paralela, apresenta-se no Capítulo 7 uma versão com suporte à elasticidade da API OpenMP do compilador GCC 4.7.

5.4 Considerações Finais

Neste capítulo, apresentou-se uma abordagem para o desenvolvimento de aplicações elásticas científicas, na qual as operações de alocação e desalocação de recursos são realizadas pela própria aplicação com base no uso de primitivas de elasticidade. Apresentou-se também o Cloudine, que implementa a abordagem proposta e oferece suporte ao desenvolvimento e execução de aplicações elásticas.

No contexto das aplicações científicas, o controle da elasticidade em nível de programação apresenta uma série de vantagens em relação às abordagens utilizadas nos mecanismos de elasticidade atuais, que em sua grande maioria se baseiam em sistemas de monitoramento ou são restritos a determinados modelos de aplicação. Dentre estas vantagens pode-se destacar: a possibilidade de se construir controladores de elasticidade sob medida para cada aplicação que considerem eventos internos, entradas de dados e parâmetros; independência de sistemas externos ou da interação com o usuário; desenvolvimento de aplicações elásticas em diversos modelos usando uma única metodologia/ferramenta; e a possibilidade de explorar a elasticidade em diferentes granularidades, permitindo a alocação de uma simples CPU até um cluster virtual completo.

Em contrapartida, a principal desvantagem da abordagem proposta está relacionada à oneração do trabalho do programador, que passa a ter que considerar as questões da elasticidade no projeto e implementação das aplicações. Outro ponto que deve ser considerado é que a inserção da lógica do controle da elasticidade no código-fonte pode gerar alguma sobrecarga (*overhead*) na execução da aplicação, como pode-se observar na Seção 6.2.

No próximo capítulo apresenta-se um conjunto de experimentos nos quais emprega-se o controle de elasticidade em nível de programação para o desenvolvimento de aplicações científicas elásticas.

CAPÍTULO 6

EMPREGANDO O CLOUDINE PARA A EXPLORAÇÃO DE ELASTICIDADE EM APLICAÇÕES CIENTÍFICAS

Neste capítulo apresenta-se um conjunto de experimentos usando o *framework* Cloudine, nos quais emprega-se a abordagem baseada no controle de elasticidade em nível de programação para o desenvolvimento de aplicações científicas elásticas.

Inicialmente, na Seção 6.1, descreve-se o ambiente computacional utilizado. Na Seção 6.2, analisa-se a sobrecarga causada pelo uso das primitivas de elasticidade. Na Seção 6.3 apresentam-se um alguns experimentos para a validação do *framework* utilizando duas aplicações sintéticas. Nas Seções 6.4 e 6.5 apresentam-se os resultados do uso do Cloudine em aplicações reais, mais especificamente em uma aplicação de montagem de genomas (SAND) e em um modelo atmosférico (OLAM).

6.1 Ambiente Computacional

Os experimentos foram executados em uma nuvem privada OpenNebula versão 3.4 com suporte a virtualização Xen. A plataforma OpenNebula originalmente não suporta elasticidade vertical, e dessa forma foram necessárias algumas alterações para incluir essa funcionalidade. O uso de OpenNebula e Xen permite utilizar todas as funcionalidades oferecidas na implementação atual do Cloudine, tornando possível a exploração de elasticidade vertical e horizontal.

O hardware utilizado é composto por um servidor equipado com três processadores AMD Opteron 6136, com 8 cores de 2.40 GHz, 98 GB RAM e 2 TB de armazenamento. Foram utilizados 4 cores e 10 GB RAM para o domínio Xen dom0 e o restante dos núcleos e da memória foram disponibilizados para o uso nas máquinas virtuais. O sistema operacional utilizado em todas as máquinas, física e virtuais, é o Ubuntu Server 12.04 com kernel 3.2.0-29.

6.2 Sobrecarga das primitivas elásticas

Nesta seção, apresenta-se a sobrecarga (*overhead*) causada por cada uma das primitivas implementadas na API de elasticidade do *framework* Cloudine. Os tempos foram coletados após 20 execuções de cada primitiva. O tempo médio e o desvio padrão de cada chamada são apresentados na Tabela 6.1.

Tabela 6.1: Sobrecarga geradas pelas primitivas de elasticidade

Função	**Tempo Médio (s)**	**Desvio Padrão**
`int clne_get_freemem()`	0,7097	0,30
`int clne_get_maxmem()`	0,7402	0,24
`int clne_get_mem()`	0,7780	0,17
`int clne_get_maxcpu()`	0,7524	0,21
`int clne_get_vcpus()`	0,0001	0,00
`int clne_get_freecpu()`	1,5121	0,33
`int clne_add_vcpu(int N)`	0,7496	0,15
`int clne_rem_vcpu(int N)`	0,7528	0,13
`int clne_add_memory(long int N)`	1,1971	0,23
`int clne_rem_memory(long int N)`	1,2036	0,19
`int clne_add_node(int N)`	276,8901	59,97
`int clne_rem_node(int N)`	0,7326	0,15

De modo geral, as primitivas `clne_get` apresentam um tempo médio muito próximo, uma vez que as implementações são muito similares, envolvendo duas comunicações via sockets (requisição e resposta) e alguma coleta de informação executada pela plataforma. São exceções as primitivas `clne_get_vcpus` e `clne_get_freecpu`. A primeira não envolve comunicação via rede, uma vez que as informações são coletadas diretamente na máquina virtual, por isso apresenta uma sobrecarga quase nula. Já a segunda necessita de um número maior de informações que devem ser coletadas pela plataforma para o cálculo do número de VCPUs livres, o que resulta em seu tempo maior.

As operações envolvendo a adição e remoção de VCPUs apresentam um tempo muito próximo ao apresentado pelas primitivas `clne_get`, pois similarmente envolvem uma única chamada ao hipervisor e duas comunicações. Por sua vez, as operações de adição e remoção de memória apresentam uma sobrecarga superior, uma vez que utilizam-se da técnica de *ballooning* do hipervisor, que possui um tempo de resposta mais significativo devido às alterações nos espaços de endereçamento da MV.

A operação envolvendo a instanciação de máquinas virtuais é a que apresenta a maior sobrecarga. A criação da máquina virtual envolve a cópia da imagem de máquina virtual e a inicialização do sistema operacional o que resulta em um grande intervalo entre a sua requisição e efetivamente estar pronta para uso. Já a destruição da MV apresenta sobrecarga menor, considerando que apenas uma confirmação da solicitação é enviada pela plataforma, uma vez que a máquina será encerrada na sequência.

6.3 Aplicações Sintéticas

Nesta seção apresenta-se um conjunto inicial de experimentos que foram realizados usando aplicações sintéticas. O objetivo destes testes é validar a abordagem de exploração da elasticidade baseada em primitivas e avaliar o funcionamento do Cloudine. São apresentados 2 conjuntos de testes, utilizando uma aplicação de transferência de calor paralelizada com OpenMP e uma aplicação de ordenação de arquivos implementada no modelo *bag-of-tasks*, respectivamente.

6.3.1 Transferência de calor *multithread*

Um problema clássico de aplicação de métodos numéricos em fenômenos físicos é a transferência de calor em uma placa plana homogênea. Considerando que todos os pontos internos da placa estejam a uma temperatura inicial diferente das temperaturas das bordas, o problema consiste em determinar a temperatura em qualquer ponto interno da placa em um dado instante de tempo [93]. A solução implementada consiste na montagem de um sistema de equações e sua resolução (via Gradiente Conjugado) para cada passo de tempo de simulação.

Com esta aplicação, realizou-se um experimento no qual empregam-se as primitivas de elasticidade para permitir que recursos ociosos na nuvem (CPUs) sejam utilizados para a redução do tempo de execução. Foi necessário modificar o código OpenMP original com primitivas de elasticidade, conforme apresentado na Figura 6.1. A aplicação possui um laço iterativo que determina a evolução temporal da simulação. No início deste laço, a primitiva `clne_get_freecpu()` foi inserida no código-fonte para verificar se existem CPUs ociosas na máquina física. Caso CPUs estejam disponíveis, elas são alocadas para a MV usando a pri-

mitiva `clne_add_vcpu()`. O número de *threads* OpenMP ativas foi definido como o mesmo número de VCPUs da MV. Além disso, foi desenvolvido uma aplicação auxiliar, com o objetivo de liberar CPUs de maneira controlada.

```
transferência_calor()
    ...
    para cada iteração:
        CPUs_livres=clne_get_freecpu()

        se CPUs_livres>0
            clne_add_vcpu(CPUs_livres)
            threads=threads+CPUs_livres

        calcula_calor_OpenMP(threads)
```

Figura 6.1: Versão elástica da aplicação de transferência de calor OpenMP.

A Figura 6.2 apresenta os resultados da alocação elástica de recursos. Foram executadas 100 iterações da aplicação de transferência de calor em um domínio quadrado com 8.192×8.192 células. No início do teste, foram atribuídas duas VCPUs para a MV da aplicação OpenMP e 18 VCPUs para a aplicação auxiliar. Na Figura 6.2, é possível observar o resultado de duas execuções usando recursos elásticos. Na primeira, uma VCPU é liberada pela aplicação auxiliar a cada 240 segundos que é alocada pela aplicação OpenMP, que termina a sua execução com 11 VCPUs. A aplicação apresenta um tempo de execução de aproximadamente 1.205 segundos, sendo 5,3 mais rápida do que a execução utilizando apenas uma VCPU (3.220 segundos).

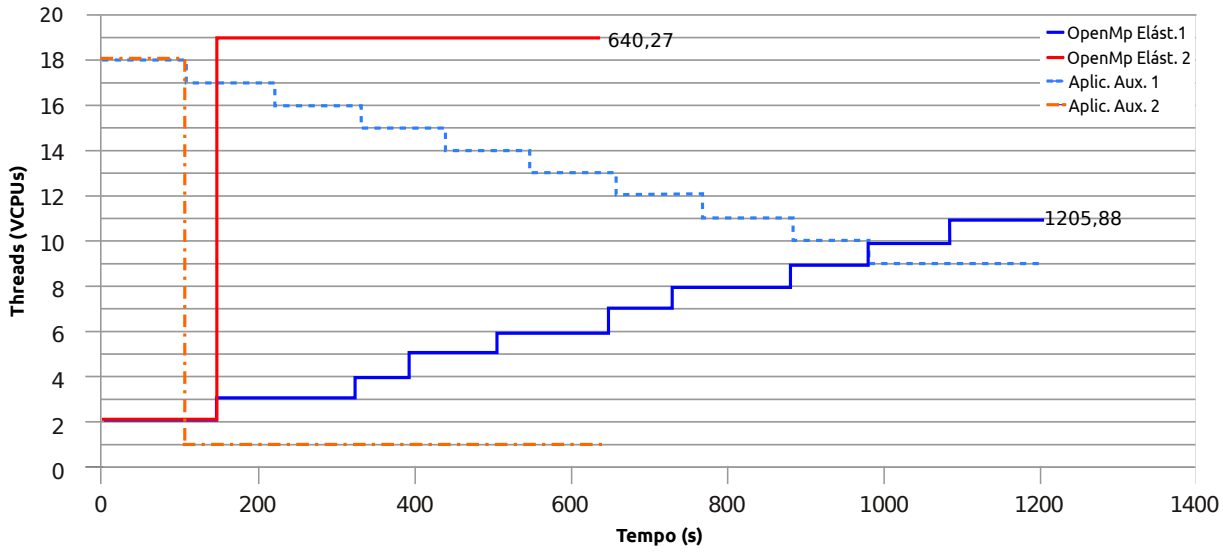

Figura 6.2: Alocação elástica de recursos ociosos.

Na segunda execução, 17 VCPUs são liberados pela aplicação auxiliar após 240 segundos. Algum tempo depois, todos os recursos são alocados pela aplicação elástica. Neste caso, o tempo de execução foi de 640,27 segundos, apresentando *speedup* de 10 vezes. Em ambos os casos, a sobrecarga gerada pelas primitivas de elasticidade foi de aproximadamente 4%.

Note que a versão não-elástica da aplicação OpenMP no mesmo cenário (com apenas 2 VCPUs disponíveis em sua inicialização) apresenta tempo médio de execução de 3.220 segundos, sendo 2,67 e 5,02 vezes mais lento do que a aplicação elástica nos dois casos apresentados. É importante observar que a versão não-elástica usando 19 *threads* apresenta tempo de execução de 429 segundos. No entanto, se considerarmos o tempo de espera para adquirir todos as 19 VCPUs necessárias para executar a aplicação, o tempo total (tempo de espera somado ao tempo de execução) seria de 2.469 segundos, considerando a liberação de uma VCPU cada 120 segundos, e 669 segundos se todos os recursos se tornassem disponíveis após 120 segundos.

Analisando ambos os cenários, é possível ver as vantagens de se empregar a elasticidade para explorar o uso de recursos ociosos. A exploração da elasticidade só foi possível por que a aplicação foi instrumentada adequadamente para solicitar e utilizar recursos adicionais. Solução similar não pode ser alcançada com os mecanismos atuais de elasticidade.

6.3.2 Ordenação de arquivos *bag-of-tasks*

Nesta seção, utilizou-se uma aplicação de ordenação de arquivos implementada no modelo *bag-of-tasks*, como ilustrado na Figura 6.3. A aplicação é constituída por um *servidor* e por *N trabalhadores*. O papel do servidor é controlar a lista de arquivos a serem classificados e criar dinamicamente os trabalhadores. Os trabalhadores obtém os arquivos via *download*, realizam a classificação, fazem o *upload* dos resultados, e por fim, enviam uma mensagem para o servidor indicando que a ordenação de arquivo está completa. A comunicação é feita via *sockets* TCP.

Essa aplicação foi usado em dois experimentos, nos quais avaliou-se: (1) o uso das primitivas de elasticidade horizontal fornecida pelo *framework* Cloudine, e (2) o emprego das primitivas de elasticidade para a adaptação da aplicação para tirar proveito dos recursos de baixo custo em um cenário que simula a utilização das *Spot Instances* oferecidas pela Amazon.

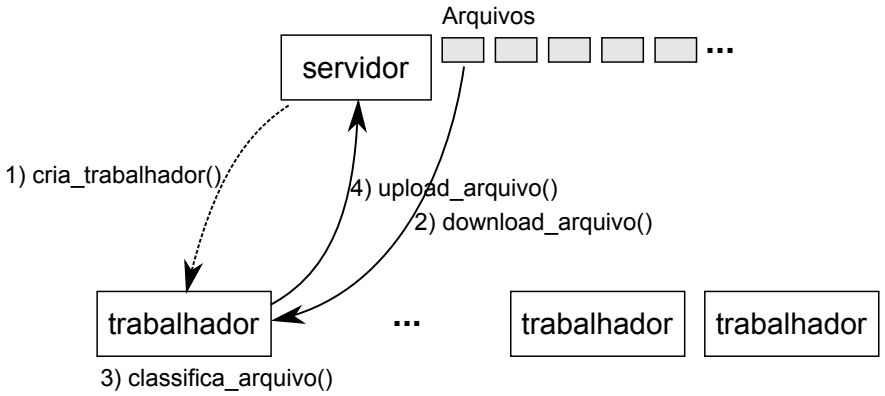

Figura 6.3: Aplicação *bag-of-tasks* para ordenação de arquivos.

No primeiro experimento, a elasticidade horizontal é empregada para a criação dinâmica de máquinas virtuais para a alocação de processos trabalhadores, usando recursos ociosos na nuvem OpenNebula. Se há recursos disponíveis, instancia-se uma nova MV para a execução de um trabalhador usando a primitiva `clne_add_node()`. A alocação de trabalhadores adicionais para quando 95% dos arquivos estiverem ordenados. Quando não há mais arquivos para processamento, os trabalhadores desalocam as suas próprias máquinas hospedeiras usando a primitiva `clne_rem_node()`. Nos testes, foram utilizados 400 arquivos com 100 mil linhas cada. A distribuição dinâmica de trabalho é apresentada na Figura 6.4. Os quadrados em branco representam o pedido de atribuição enviado pelo servidor e os quadrados pretos representam o momento em que a máquina foi efetivamente alocada e o trabalhador inicia o sua operação.

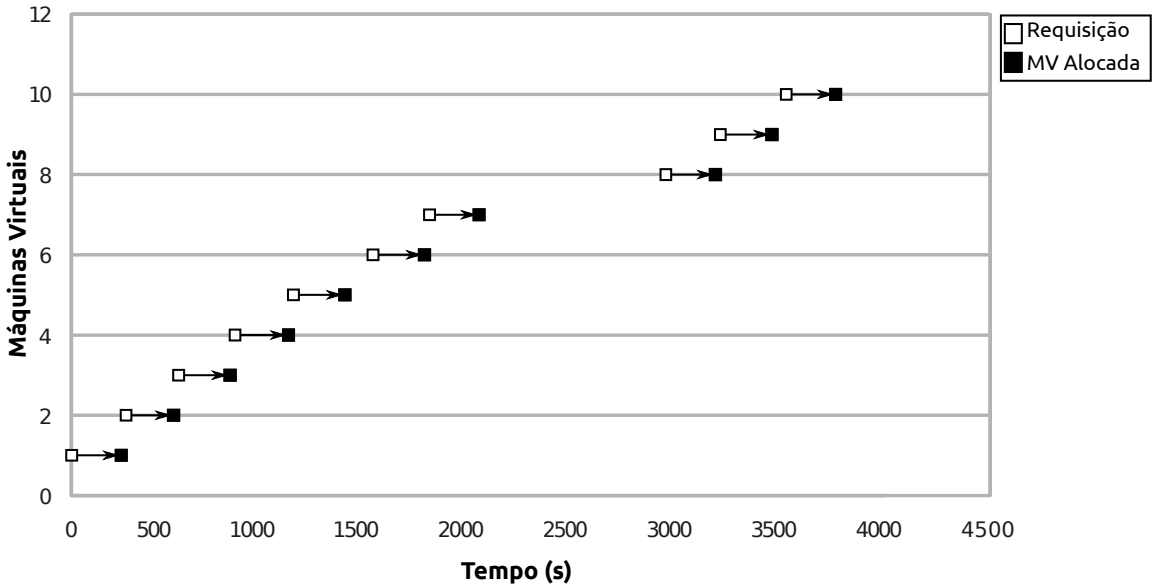

Figura 6.4: Alocação dinâmica de máquinas virtais.

No início da execução, 7 MVs são alocadas uma após a outra aproveitando recursos ociosos. Depois de aproximadamente 900 segundos, novos recursos são liberados e as últimas três máquinas são alocadas. O experimento termina após 4.500 segundos, com 10 nodos trabalhadores. O tempo de inicialização das MVs foi em média 4 minutos.

No segundo experimento, o objetivo é usar primitivas de elasticidade para permitir que as aplicações controlem a alocação de recursos com base na disponibilidade de recursos de baixo custo. Para testar esta funcionalidade em um ambiente controlado (e permitir a replicação dos testes), simulou-se o comportamento de *Spot Instances* na nuvem OpenNebula.

Para utilizar as *Spot Instances*, o usuário especifica o tipo e a quantidade de instâncias que deseja alocar e dá um lance com o valor máximo (*bid-value*) que está disposto a pagar por hora de uso de cada instância. Assim, se o lance do usuário for maior ou igual ao valor estipulado pela Amazon, a solicitação será atendida. As instâncias alocadas ficarão disponíveis para o usuário até que este as finalize ou até que o preço da instância supere o lance dado pelas alocações (o que acontecer primeiro). Se uma *Spot Instance* é interrompida pela Amazon, não se cobra pelo uso parcial dos recursos.

Para lidar com o término repentino das máquinas virtuais, o servidor foi modificado para colocar de volta na fila de trabalhos todos os arquivos que estavam sendo processados naquele momento. Para simular o funcionamento das *Spot Instances*, implementou-se um controlador externo à aplicação que encerra as MVs quando o valor (*Spot Price*) excede a oferta do usuário e notifica o servidor quando a alocação de MV volta a estar disponível. Enquanto o valor for superior à oferta do usuário, nenhuma MV pode ser alocada.

Para determinar a interrupção das máquinas, utilizou-se o histórico de preços do dia 29 de maio de 2013 para as instâncias do tipo *m1.large* hospedadas na zona *us-west-2b*, conforme ilustrado na Figura 6.5. O período simulado inicia-se às 2:00 e termina às 7:00.

O objetivo é investir no máximo $\$1,00$ por hora, considerando que na simulação, o servidor é alocado em uma instância sob demanda do tipo *Large* a um custo de $\$0,24$ por hora. Assim, definiu-se o valor do lance (*bid value*) em $\$0,045$, permitindo a alocação do servidor e 16 trabalhadores simultaneamente ($\$0,96$/hora). Neste teste, utilizou-se uma carga de trabalho constituída por 1.000 arquivos com 100 mil linhas cada. A Figura 6.6 mostra os resultados.

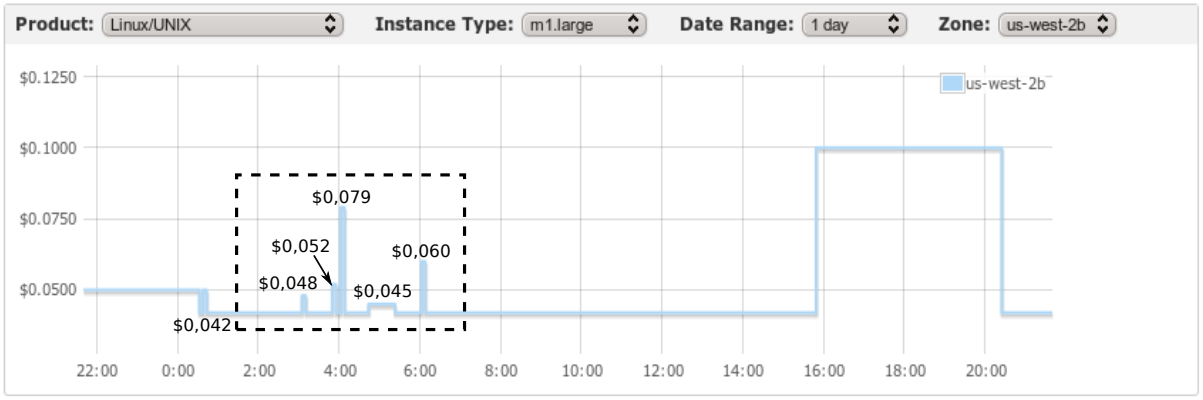

Figura 6.5: Histórico de preços das *Spot Instances*.

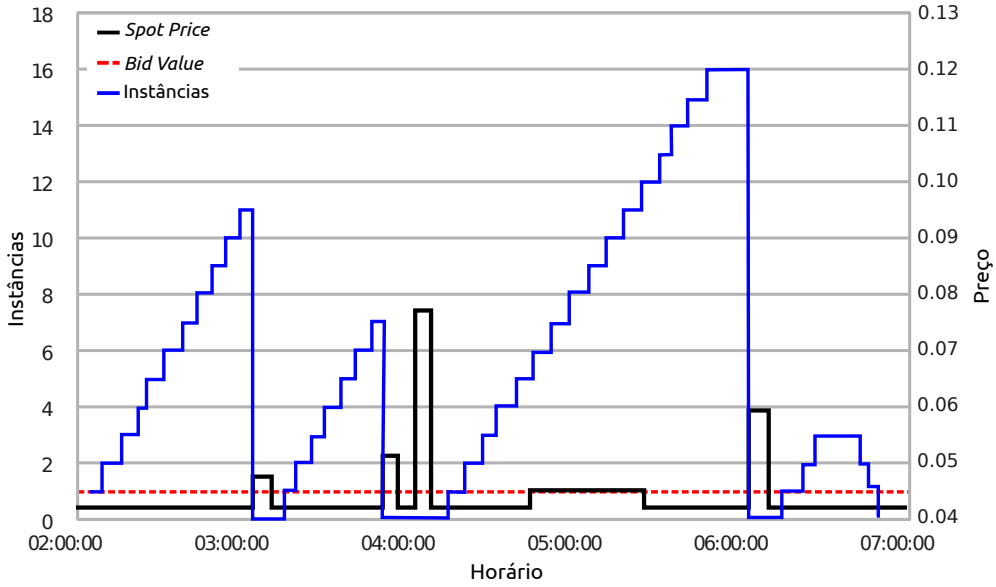

Figura 6.6: Alocação de máquinas virtuais na simulação de *Spot Instances*.

A simulação começa às 2:00, quando o servidor é iniciado. Gradualmente, 11 trabalhadores são alocados um a um até 3:04, quando o *Spot Price* excede o valor de compra e todas as máquinas virtuais são finalizadas. Seis minutos depois, há uma redução de preços e a alocação de máquinas é reiniciada. Sete máquinas são alocadas até que o preço aumenta novamente às 3:49. Às 04:07 os preços voltam a cair e permanecem baixos por um longo período, permitindo a alocação de 16 máquinas virtuais que são encerradas às 6:02. Finalmente, às 6:08, o servidor aloca três máquinas que são usadas para concluir o processamento dos arquivos 40 minutos depois. O valor total gasto para executar a aplicação foi de aproximadamente $\$1,75$, considerando que as MVs foram encerradas três vezes e as horas parciais de uso não foram cobradas. Neste experimento, o tempo de inicialização das MVs foi em média de 6 minutos.

Os resultados apresentados em ambos experimentos mostram que o controle da elasticidade em nível de programação pode ser utilizado também em aplicações distribuídas, fornecendo funcionalidades que não podem ser fornecidas pelos mecanismos tradicionais de elasticidade.

6.4 *Scalable Assembler at Notre Dame* - SAND

Neste experimento, objetiva-se demonstrar que a abordagem de exploração de elasticidade implementada pelo Cloudine é bastante flexível e genérica, permitindo que seja combinada com outros *frameworks* de elasticidade. Neste caso particular, o Cloudine foi empregado em conjunto com o *framework* Work Queue [116] na adaptação de uma aplicação elástica de montagem de genomas - SAND.

O SAND é um conjunto de módulos para a montagem do genoma construídos para a execução em ambientes de computação distribuídos, tais como clusters, grades e nuvens [104].

A aplicação é composta por duas fases principais: seleção de candidatos e alinhamento de sequências. A etapa de seleção de candidatos sugere pares de leituras que podem se sobrepor. Toma-se como entrada um conjunto de sequências e gera-se um conjunto de pares de candidatos para a fase de alinhamento. A fase de alinhamento utiliza como entrada o conjunto de sequências e uma lista de pares de sequências candidatas criados na etapa anterior, e gera como saída um conjunto de registros de sobreposição indicando quais sequências alinham-se adequadamente. Este conjunto de pares é utilizado no estágio final do montador de genomas.

Ambas as etapas são implementadas como uma aplicação mestre-escravo usando o *framework* Work Queue. Neste *framework*, o processo mestre é responsável por enviar um módulo executável (binário) e os arquivos de entrada para os escravos. Os escravos invocam o módulo executável para processar os dados enviados, armazenando os resultados localmente na máquina onde são executados. Quando a tarefa é concluída, os resultados são enviados ao mestre, que verifica o resultado e o armazena permanentemente. Em sua especificação original, o mestre deve ser instanciado manualmente em uma máquina e os trabalhadores podem ser executados em nós de um cluster, grade ou nuvem, de modo manual ou usando um sistema de lote, como o Condor[1].

[1] `http://www.cs.wisc.edu/condor`

Neste experimento, o processo mestre de ambas as etapas foi modificado para alocar de modo automático novos escravos usando recursos da nuvem OpenNebula, como apresentado no pseudo-algoritmo da Figura 6.7. O processo mestre verifica constantemente se existem recursos disponíveis para a criação de novas MVs para executar os escravos. Em caso positivo, uma nova MV é criada e o próprio mestre inicia o trabalhador na máquina. Quando a tarefa é concluída, todas as MVs utilizadas pelos processos escravos são finalizadas.

```
mestre_SAND()
    ...
    enquanto(1):
        se recursos_livres
            clne_add_node(1)
            //aguarda a inicializacao da MV

        upload_trabalhador_MV
        inicia_trabalhador_MV
    ...
```

Figura 6.7: Alocação de MVs e inicialização de trabalhadores pelo processo mestre.

A Figura 6.8 apresenta uma linha do tempo da alocação das máquinas virtuais utilizadas no processamento do genoma do mosquito *Anopheles gambiae Mopti*[2], na qual os círculos representam a criação das MVs, os quadrados representam a desalocação das MVs, e as estrelas representam o início/fim das fases de processamento.

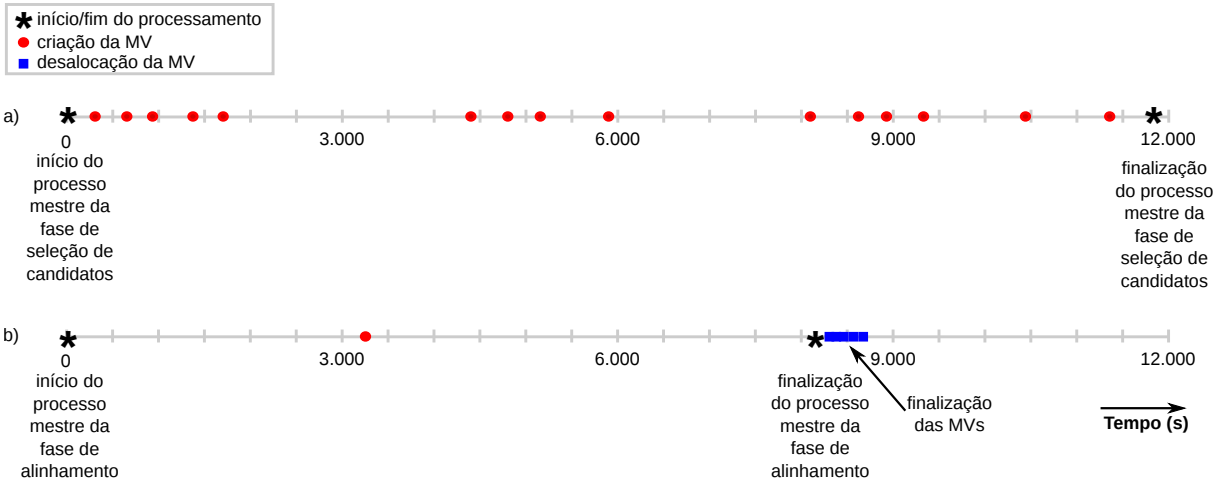

Figura 6.8: SAND: Alocação dinâmica de máquinas virtuais.

No início da execução (Figura 6.8a), o processo mestre da fase de seleção de candidatos é iniciado e 15 máquinas virtuais são criadas para hospedar processos escravos. A etapa de

[2]Disponível em `http://www3.nd.edu/~ccl/software/sand/`

seleção é finalizada após 11.800 segundos. Na sequência, a fase de alinhamento é iniciada (Figura 6.8b). As 15 MVs anteriormente criadas são também utilizadas nesta fase e um nova máquina é instanciada após 3.300 segundos. Ao término do processamento, todas as MVs são finalizadas e os recursos são liberados. O *speedup* alcançado na fase de alinhamento foi de 12 vezes (se comparado à execução usando um processo escravo), de acordo com os dados apresentados na interface da aplicação.

Como os resultados mostram, o uso do *framework* Cloudine mostrou-se eficaz na construção dos mecanismos para provisionamento automático de escravos para o SAND. O mecanismo implementado permite que o usuário inicie a aplicação utilizando o mínimo de recursos, e que esta aloque posteriormente seus próprios recursos sem a necessidade da interação do usuário ou o uso de escalonadores (como originalmente especificado).

Note que a política de alocação de recursos adicionais neste experimento foi baseada na disponibilidade de recursos, mas pode ser modificada de acordo com condições desejadas. Por exemplo, é possível alocar recursos seguindo uma função custo, ou ainda, de acordo com uma restrição de tempo de resposta (*deadline*).

6.5 *Ocean-Land-Atmosphere Model* - OLAM

Nesta seção, utiliza-se o *framework* Cloudine para adicionar suporte à elasticidade ao modelo OLAM. A elasticidade é usada para implementar a alocação dinâmica de processadores virtuais e memória, bem como na implementação de balanceamento de carga, não disponível na versão original do modelo. O objetivo dos experimentos é demonstrar que o uso da elasticidade pode agregar novas funcionalidades à aplicação, de modo a tirar proveito das características da computação em nuvem.

O OLAM é um modelo de simulação numérica para a climatologia desenvolvido pela universidade norte-americana Duke [148]. O modelo consiste essencialmente em uma representação de volumes finitos das equações de Navier-Stokes não-hidrostáticos sobre a atmosfera do planeta com uma formulação de leis de conservação de massa, quantidade de movimento, temperatura e operadores numéricos que incluem a divisão de tempo para termos acústicos [98]. Os volumes finitos são definidos horizontalmente por uma malha global com células triangulares

que subdivide-se verticalmente através da altura da atmosfera, formando prismas verticalmente empilhados de bases triangulares.

O OLAM foi originalmente desenvolvido em Fortran 90 e paralelizado usando MPI, no modelo de programação Programa Único, Múltiplos Dados (*Single Program, Multiple Data - SPMD*). Neste trabalho, utilizou-se uma versão re-implementada usando a linguagem C e paralelizada com MPI, para paralelismo inter-nodos, e OpenMP para paralelismo intra-nodo [126]. Nesta versão também incluiu-se refinamento de malha em tempo de execução (*Online Mesh Refinement* - OMR) [125]. A implementação de OMR permite refinar a malha em tempo de execução, aumentando a resolução da representação discreta em partes específicas do domínio descritas no código-fonte.

A execução do OLAM envolve várias etapas e pode ser dividida em três partes principais: inicialização, cálculo atmosférico e armazenamento de resultados, conforme ilustrado na Figura 6.9. A primeira parte do código envolve o pré-processamento, onde as configurações são lidas, a memória para as estruturas de dados são alocadas, e as informações do terreno, vegetação, solo e mar são processadas.

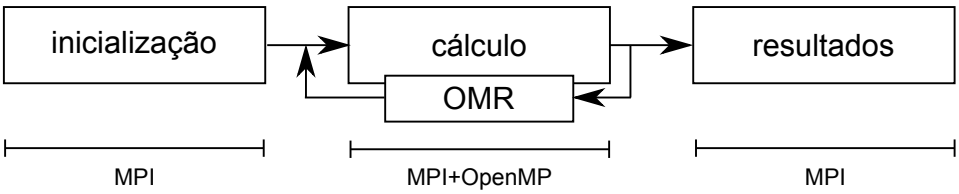

Figura 6.9: Etapas de execução do OLAM.

O cálculo atmosférico consiste em uma etapa iterativa, na qual são calculadas a transferência de radiação, micro-física, sistemas biofísicos, turbulência e posicionamento de nuvens convectivas (cumulus). Durante a etapa iterativa é possível refinar a malha. Neste caso, a execução é interrompida e a malha é refinada em pontos específicos da Terra, de acordo com uma condição climatológica. Após a conclusão do refinamento de malha, a execução iterativa prossegue normalmente. Ao término da etapa iterativa, os resultados da simulação são armazenados em disco.

Todas as etapas do modelo são paralelizadas usando MPI. O OpenMP é usado apenas na parte iterativa para executar os laços da função *progwrtu*, responsável por até 70% do tempo de CPU de cada iteração (passo de tempo).

Considerando que apenas a parte iterativa é paralelizada usando OpenMP, é possível usar a elasticidade para permitir o uso eficiente de recursos, de modo que os recursos sejam alocados somente quando exigidos e evitando que estes fiquem ociosos nas fases de inicialização e escrita de resultados. Para tal, empregou-se as primitivas de elasticidade para permitir que a execução do OLAM inicie com uma única VCPU e novos processadores sejam alocados dinamicamente no passo iterativo, onde eles são realmente utilizados pelas *threads* OpenMP, como apresenta-se na Figura 6.10.

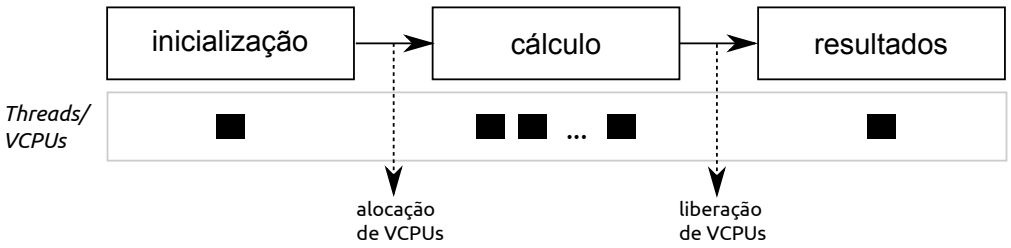

Figura 6.10: Alocação dinâmica de VCPUs na fase iterativa do OLAM.

Para testar a alocação dinâmica de VCPUs, foram executadas 1.440 iterações antes do OMR e 1.440 após, simulando 2 dias de tempo real (cada iteração representa 60 segundos de tempo real). A malha empregada tem resolução global de 50 km, e o seu refinamento é feito a partir da coordenada Cartesiana 0,0 com raio de 2.000 km. Foram usadas duas configurações de cluster virtuais, com dois e quatro nodos de processamento.

Na primeira configuração, os dois nodos possuem inicialmente uma única VCPU e outras sete são adicionadas na primeira fase iterativa, liberados na fase de OMR e realocados na próxima fase iterativa, de acordo com a demanda das regiões paralelas criadas por OpenMP. O tempo total de execução é de 4.396 segundos, cerca de 2% mais lento do que a execução usando oito VCPUs por processo durante toda a execução (4.304 segundos). O processo de alocação e desalocação é ilustrado na Figura 6.11.

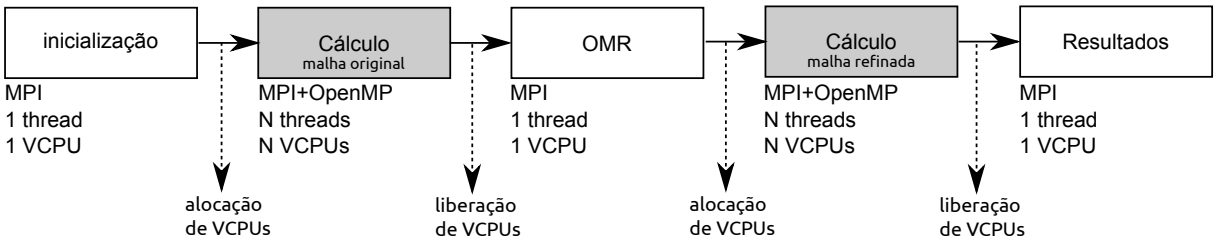

Figura 6.11: Alocação dinâmica de VCPUs.

Da mesma forma, usando a segunda configuração, a execução começa com quatro nós com uma VCPU e as demais são alocadas dinamicamente para as etapas de cálculo iterativo. O tempo total de execução é de 4.016 segundos, cerca de 1,7% mais lento do que a execução com quatro VCPUs por processo MPI durante toda a execução (3.950 segundos).

Os resultados mostram que é possível empregar com sucesso a elasticidade para alocar recursos de forma eficiente, melhorando o uso compartilhado da infraestruturas da nuvem com sobrecarga (*overhead*) pouco significativa.

As primitivas de elasticidade também foram usadas para fornecer balanceamento de carga para o OLAM, originalmente não implementada no modelo. Essa funcionalidade se faz necessária devido ao fato de que o refinamento de malha em tempo de execução causa desbalanceamento de carga, uma vez que os novos elementos da malha criados não são distribuídos uniformemente entre todos os processos MPI. A Tabela 6.2 mostra a distribuição de carga antes e depois do OMR, em um simulação usando a malha de 50 km de resolução e quatro processos. É possível observar que os processos P2 e P3 são significativamente afetados pelo refinamento da malha, apresentando, respectivamente, 75% e 68% mais elementos tomando como base o processo P0.

Tabela 6.2: Desbalanceamento de carga causado pelo refinamento da malha.

Processo	**Elementos antes do OMR**	**Elementos após OMR**	**Diferença em relação à P0**
P0	50.556	50.556	–
P1	50.812	50.812	0,5%
P2	51.110	88.875	75%
P3	50.886	85.143	68%

Diferentemente das técnicas tradicionais de balanceamento de carga, que são baseadas na redistribuição da carga entre os processos, no método proposto distribui-se automaticamente

as VCPUs disponíveis (16 no experimento) proporcionalmente à carga de trabalho de cada processo MPI. No cenário apresentado na Tabela 6.2, quatro VCPUs são alocadas para cada processo MPI antes do OMR. Após o refinamento, o balanceador proposto redistribui as VCPUs entre os processos, alocando 3 VCPUs para os processos P0 e P1 e 5 VCPUs para P2 e P3, de modo a equilibrar a relação carga/VCPU.

Na Figura 6.12 compara-se o tempo médio de iteração obtida com e sem balanceamento de carga. Pode-se observar que o balanceamento de carga reduziu o tempo de processamento das iterações (não considerando as trocas de mensagens MPI) em 24%, em média. Esse resultado representa uma redução de 17% no tempo de execução da etapa de cálculo (de 1.999 para 1.720 segundos) e 7% do tempo total de execução do modelo (3.950 a 3.677 segundos). Note que foi possível obter ganhos de desempenho com a mesma quantidade de recursos.

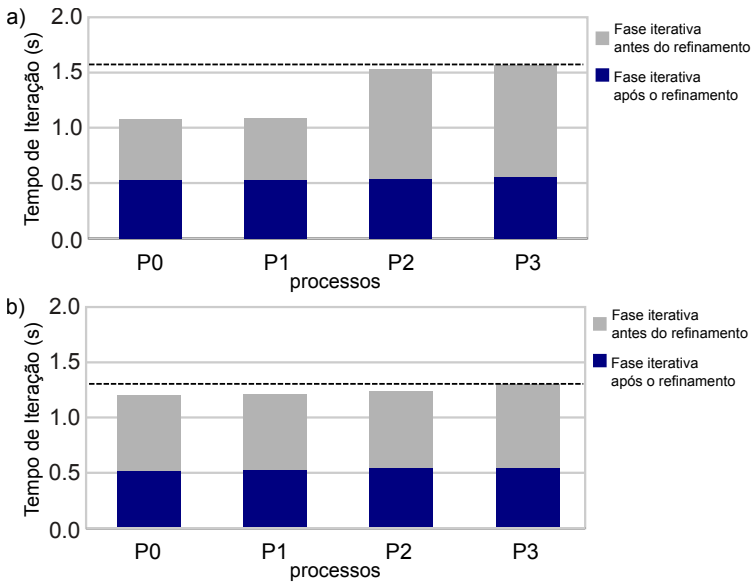

Figura 6.12: Tempo de execução da iteração. a) Sem redistribuição dos recursos. b) Com redistribuição dos recursos.

O uso das primitivas oferecidas pelo Cloudine na construção do balanceador de carga para o OLAM mostrou-se ser uma opção interessante. Em comparação com as estratégias tradicionais de balanceamento de carga, a abordagem proposta exige poucas modificações no código fonte original (algumas dezenas de linhas) e não exige a interrupção da aplicação ou a redistribuição da carga entre os processos, o que geralmente envolve trocas de dados e resulta em sobrecargas elevadas [55]. De acordo com o estado da arte, não há relato do uso da elasticidade para fornecer balanceamento de carga como o apresentado nesta seção.

Por fim, também adicionou-se suporte à alocação elástica de memória. A capacidade de modificar dinamicamente a memória de uma MV em tempo de execução, sem qualquer interrupção do serviço, representa um recurso importante para aplicações com requisitos de memória dinâmicos.

Considerando que o OLAM pode ser executado com diferentes resoluções e refinamentos, escolher a quantidade mais adequada de memória não é uma tarefa trivial. Assim, implementou-se um mecanismo para alocação de memória para a máquina virtual de acordo com as exigências da aplicação. O mecanismo foi projetado para alocar a memória necessária para a aplicação e para manter livre a quantidade inicial de memória MV (para ser usado pelo sistema operacional ou outros serviços). Esta funcionalidade também é útil nos casos em que o OMR requer a alocação de espaço adicional para o armazenamento dos novos elementos de malha.

A alocação dinâmica de memória foi implementada através da substituição das chamadas à função `malloc()`[3] por chamadas à função `elast_alloc()`. Esta função se assemelha ao `malloc` mas também adiciona à MV o espaço de memória solicitado usando chamadas à primitiva `clne_add_memory()`, caso necessário. A alocação de memória adicional ocorre quando há menos de 512 MB livres na memória da MV. Assim, quando uma estrutura de dados é alocada, o espaço de memória MV aumenta automaticamente, evitando a ocorrência de *thrashing* [102] (que impacta consideravelmente no desempenho das aplicações) e impedindo a aplicação de abortar devido à falta de memória.

Testou-se a alocação dinâmica de memória em uma simulação de 120 iterações para cada fase iterativa (antes e depois da OMR) simulando 4 horas de tempo real. Empregou-se uma malha com resolução global de 40 km. Foram utilizadas duas máquinas virtuais com 2 GB de memória. Figura 6.13 mostra a alocação de memória para os processos P0 e P1.

O mecanismo proposto mostrou-se eficiente na alocação de memória, mantendo os dados da aplicação em memória residente e preservando a memória para o sistema operacional e outros serviços em execução na máquina. Note que o pico de utilização ($\sim$ 55 minutos) é cerca de 8 vezes maior do que no início da simulação. Esta grande variação torna difícil prever a memória a ser alocada na instanciação da MV, o que torna interessante o uso da elasticidade.

[3] A função `malloc` é usada para alocar uma certa quantidade de memória durante a execução de um programa.

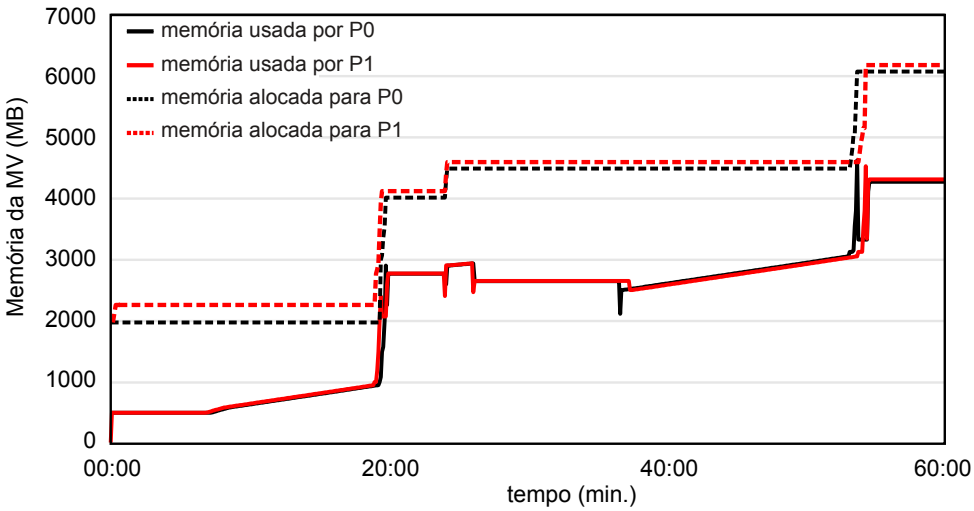

Figura 6.13: Alocação dinâmica de memória no OLAM.

Os resultados apresentados foram comparados aos resultados obtidos por um mecanismo baseado no monitoramento proposto no trabalho de Moltó et al. [102]. Esse mecanismo não obteve sucesso na alocação de memória para o pico de carga de trabalho e a aplicação foi abortada depois de aproximadamente 20 minutos de simulação, uma vez que o mecanismo não foi capaz de alocar a memória de acordo com a demanda da aplicação.

6.6 Considerações Finais

Neste capítulo apresentou-se um conjunto de experimentos realizados com aplicações desenvolvidas ou adaptadas com o Cloudine. Os resultados apresentados comprovam que o controle de elasticidade em nível de programação é uma alternativa eficiente e eficaz para o desenvolvimento de aplicações científicas elásticas. Foram desenvolvidas aplicações elásticas em diferentes modelos com funcionalidades que não poderiam ser agregadas caso mecanismos de elasticidade tradicionais tivessem sido empregados. Isso só é possível porque as particularidades das aplicações podem ser consideradas na construção de controladores de elasticidade sob medida para cada uma delas.

No próximo capítulo, apresenta-se uma versão elástica da API OpenMP do GCC. A API foi modificada para fornecer alocação elástica de recursos de modo automático e transparente na construção de aplicações *multithread*. Um conjunto de experimentos usando o OpenMP elástico também é apresentado.

CAPÍTULO 7

FORNECENDO ELASTICIDADE PARA APLICAÇÕES OPENMP

Conforme apresentado no Capítulo 5, há duas maneiras para se empregar o *framework* Cloudine. Na primeira, utiliza-se as primitivas diretamente no código-fonte para a construção de controladores de elasticidade embutidos na própria aplicação. Os resultados desta abordagem podem ser observados no Capítulo 6. Por sua vez, a segunda abordagem considera o uso das primitivas para adicionar suporte à elasticidade a bibliotecas e *frameworks* já consolidados. Dessa forma, pode-se construir aplicações elásticas de modo mais abstrato, no qual se esconde alguns detalhes do controle da elasticidade do programador mas ainda considera-se as particularidades da aplicação na alocação dos recursos. Para demonstrar o uso desta abordagem, neste capítulo apresenta-se uma versão da API OpenMP modificada para suportar a alocação dinâmica de recursos.

7.1 OpenMP

O padrão OpenMP é desenvolvido e mantido pelo grupo *OpenMP Architecture Review Board*, formado pelos maiores fabricantes de software e hardware do mundo, tais como SUN Microsystems, SGI, IBM, Intel, que no final de 1997, reuniram esforços para criar um padrão de programação paralela para arquiteturas de memória compartilhada [42]. O OpenMP consiste em uma API e um conjunto de diretivas que permitem a criação de programas paralelos com compartilhamento de memória através da implementação automática e otimizada de um conjunto de *threads*. Atualmente, o OpenMP é o padrão *de facto* de programação paralela em C/C++ e Fortran [43].

As principais diretivas OpenMP permitem que sejam criados grupos de *threads* para executar em paralelo uma região especifica de código e para dividir o trabalho de laços de repetição ou de segmentos de código. As rotinas em nível de usuário são utilizadas para configurar e recuperar informações sobre o ambiente de execução (*runtime environment*). As variáveis de

ambiente também são usadas para ajustar as configurações de aplicações OpenMP em tempo de execução.

O OpenMP baseia-se no modelo de *fork-join* [43]. Um programa OpenMP sempre começa a execução como uma única *thread* (a *thread* mestre) e quando uma diretiva *parallel* é encontrada, cria-se uma região paralela que é executada por um grupo de *threads*. Quando a região paralela é concluída, as *threads* são sincronizadas em uma barreira implícita e a *thread* mestre continua a execução. Dentro das regiões paralelas, diretivas de compartilhamento de trabalho podem ser empregadas para distribuir o trabalho entre as *threads* disponíveis. Algumas construções de compartilhamento especificam regiões de código que devem ser executadas por uma única *thread* (*omp single*, *omp master*), enquanto que outras distribuem o trabalho entre o grupo de *threads* (*omp for*) ou parte delas (*omp sections*).

Este modelo de execução, torna as aplicações OpenMP boas candidatas para uso da elasticidade, uma vez que o número de *threads* e, consequentemente, os recursos efetivamente utilizados podem variar significativamente durante a execução da aplicação. A Figura 7.1 mostra um exemplo onde a elasticidade é utilizada para fornecer recursos dinamicamente para uma aplicação OpenMP conforme ocorre a criação e a sincronização das *threads*.

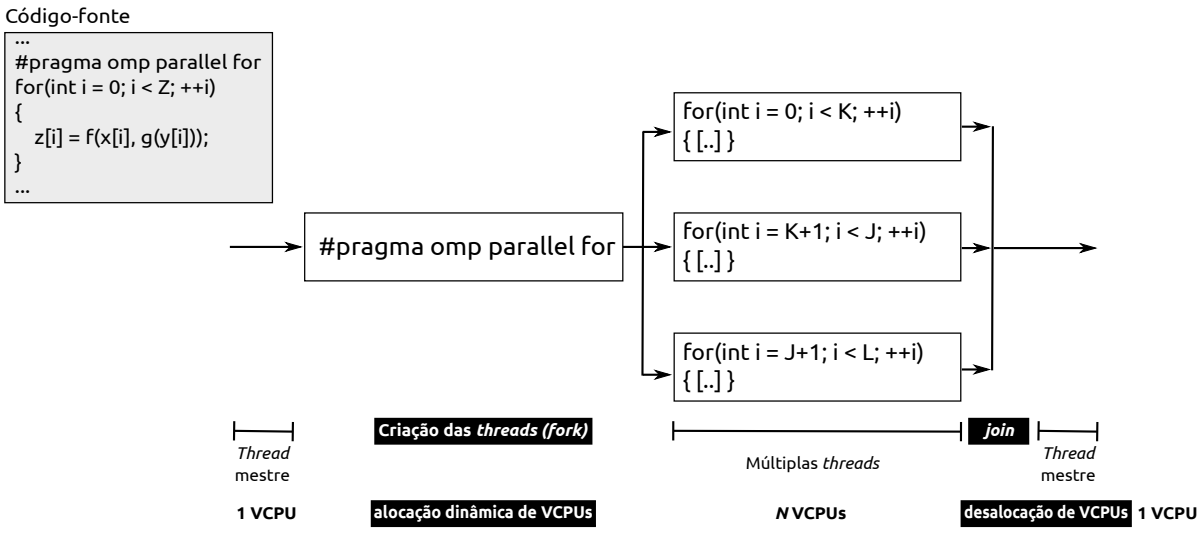

Figura 7.1: Modelo de execução *fork-join*.

Neste exemplo, um laço de repetição (*for*) é paralelizado com OpenMP por meio da diretiva *parallel for*. Inicialmente, a aplicação é executada pela *thread* mestre, que utiliza um único processador. Ao serem criadas na região paralela, as demais *threads* demandam mais recursos de

processamento, utilizando uma quantidade maior de VCPUs. Ao término da região paralela, as *threads* sincronizam-se e a aplicação volta a utilizar uma única VCPU. A existência de regiões paralelas entre regiões sequenciais permite que a alocação de múltiplas VCPUs para a máquina virtual possa ser feita apenas no momento em que forem realmente necessárias, evitando a alocação desnecessária de recursos e seus custos associados. Esse fato motivou o desenvolvimento da versão elástica para a API OpenMP.

7.2 OpenMP Elástico

Nesta seção, descreve-se o desenvolvimento da versão elástica da API OpenMP. Nesta versão, o conjunto de diretivas é estendido para suportar o ajuste automático do número de VCPUs de acordo com o número de *threads* em execução. Dessa forma, é possível esconder do programador a complexidade da escrita e execução de estratégias de elasticidade, uma vez que a própria API controla automaticamente a alocação dos recursos.

As modificações foram feitas sobre a API OpenMP fornecida pelo GCC 4.7, implementada na biblioteca *libgomp*[1]. O suporte à elasticidade foi implementado usando primitivas do *framework* Cloudine. As extensões propostas foram baseadas na adaptação de três diretivas OpenMP (*parallel*, *single* e *sections*), e na adição de um conjunto de rotinas de nível de usuário, visando fornecer um controle mais preciso sobre a execução elástica. Adicionou-se também rotinas para alocação dinâmica de memória.

A primeira diretiva a ser modificada foi *parallel*, uma vez que é base para todas as demais. Esta construção é responsável por criar um grupo de *threads* para executar a região paralela associada, que é o código contido dentro da construção. As partes do programa que não são delimitados por uma diretiva *parallel* são executadas sequencialmente [43]. No OpenMP elástico, a diretiva foi modificada para adaptar o número de VCPUs de acordo com o número de *threads* especificados na região paralela (via rotina ou cláusula).

Na Figura 7.2 apresenta-se o comportamento da diretiva *parallel* elástica. Quando a região paralela e as *threads* são criadas, VCPUs adicionais são automaticamente alocadas. Ao término da região paralela o número de VCPUs retorna à configuração anterior.

[1] `http://gcc.gnu.org/onlinedocs/libgomp/`

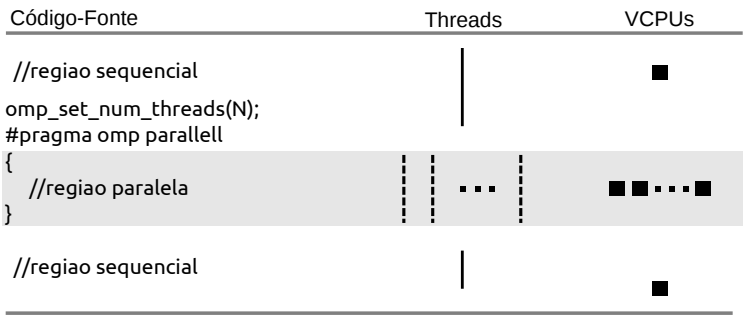

Figura 7.2: Comportamento elástico na diretiva *parallel*.

Dentro de uma região paralela, outras diretivas de compartilhamento de trabalho podem ser usadas para distribuir a computação entre as *threads* de um grupo. Existem três principais diretivas de compartilhamento de trabalho: *for*, *sections* e *single*.

A diretiva *for* é usada para dividir iterações do laço de repetição e atribuí-las a *threads* para a execução concorrente. Esta diretiva não foi modificada, uma vez que ela herda automaticamente o grupo de *threads* definido na seção paralela e as VCPUs alocadas.

A diretiva *single* identifica uma seção de código que deve ser executado por uma única *thread*, normalmente o primeiro a encontrá-la. As demais *threads* esperam em uma barreira até que a *thread* em execução do bloco de código único seja concluída. A diretiva *single* elástica mantém apenas um VCPU ativa durante a execução do bloco. Quando as *threads* são sincronizadas na barreira, o número de VCPUs é restabelecida, conforme apresenta-se na Figura 7.3.

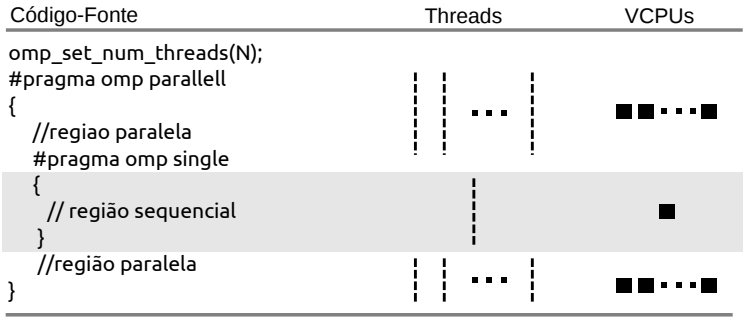

Figura 7.3: Comportamento elástico na diretiva *single*.

A diretiva *sections* fornece um meio pelo qual as diferentes *threads* podem executar diferentes blocos do programa em paralelo, porém de modo independente. No OpenMP original, cada *thread* executa um bloco de código de cada vez, e cada bloco de código será executado exatamente uma vez. Se houver menos *threads* que blocos de código, algumas *threads* executarão vários blocos de código. Se houver menos blocos de código do que *threads*, algumas delas

ficarão ociosas (determinadas pelo escalonador do OpenMP).

Por sua vez, na versão elástica da diretiva, o número de VCPUs é ajustado para o número de seções no código fonte, conforme ilustrado na Figura 7.4. O objetivo é evitar VCPUs inativas caso existem *threads* não utilizadas. De acordo com a especificação OpenMP, o número máximo de segmentos que podem ser utilizados em seções é definido pela região paralela. Assim, se existem mais seções do que *threads*, não serão adicionadas novas VCPUs.

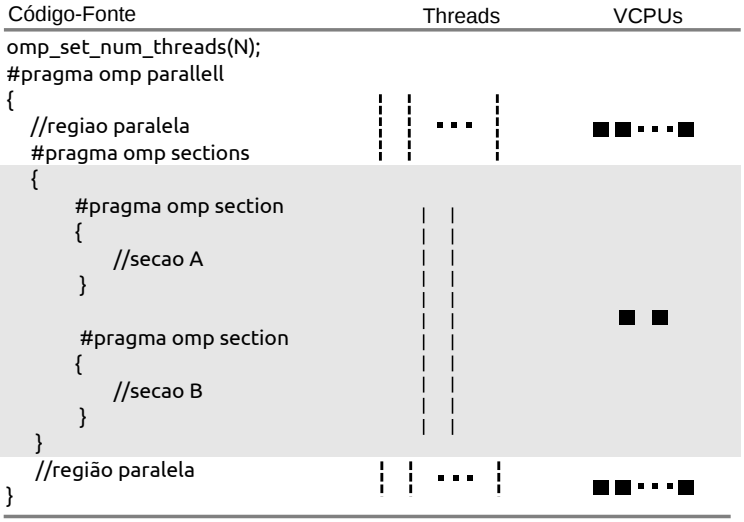

Figura 7.4: Comportamento elástico na diretiva *sections*.

É importante observar que outras diretivas OpenMP e cláusulas não são afetadas pelas extensões propostas e continuam trabalhando como proposto na especificação original.

Além das modificações nas diretrizes OpenMP, a extensão também adiciona algumas rotinas na biblioteca de nível de usuário com o objetivo de permitir aos programadores controlar a execução elástica e consultar o ambiente de execução. As rotinas adicionais são listadas e descritas na Tabela 7.1. As três primeiras rotinas listadas são usadas para habilitar e desabilitar a elasticidade fornecida pelas diretivas OpenMP, permitindo o uso da elasticidade somente quando necessário. Quando desabilitadas, as diretivas se comportam conforme previsto na especificação original. Há também três rotinas para verificar o estado das diretivas elásticas.

A rotina *omp_set_map* permite que o programador defina o número de VCPUs que devem ser alocadas em função das *threads* ativas. O argumento *m* especifica a relação entre VCPUs e *threads*. Por exemplo, o valor de 0,5 para *m* significa que para cada duas *threads* uma VCPU será alocada. Valores não inteiros são arredondados para cima.

Tabela 7.1: Rotinas adicionadas à API OpenMP.

Rotina	**Descrição**
`void omp_set_elastic_parallel(int flag)`	Habilita ou desabilita a elasticidade da diretiva *parallel*. Se o argumento *flag* é verdadeiro, o ajuste dinâmico de recursos é habilitado, se falso o ajuste é desabilitado.
`void omp_set_elastic_section(int flag)`	Habilita ou desabilita a elasticidade da diretiva *sections*. Se o argumento *flag* é verdadeiro, o ajuste dinâmico de recursos é habilitado, se falso o ajuste é desabilitado.
`void omp_set_elastic_single(int flag)`	Habilita ou desabilita a elasticidade da diretiva *single*. Se o argumento *flag* é verdadeiro, o ajuste dinâmico de recursos é habilitado, se falso o ajuste é desabilitado.
`int omp_get_elastic_parallel( )`	Retorna verdadeiro se a elasticidade está habilitada para a diretiva *parallel*. Se desabilitada, retorna falso.
`int omp_get_elastic_section( )`	Retorna verdadeiro se a elasticidade está habilitada para a diretiva *sections*. Se desabilitada, retorna falso.
`int omp_get_elastic_single( )`	Retorna verdadeiro se a elasticidade está habilitada para a diretiva *single*. Se desabilitada, retorna falso.
`void omp_set_map(float m)`	Configura a razão entre o número de VCPUs e *threads* ativas.
`int omp_get_max_cpu( )`	Retorna o número total de CPUs da máquina física onde a MV está sendo executada.
`int omp_get_free_cpu( )`	Retorna o número CPUs da máquina física disponíveis para alocação.
`void* omp_alloc(long int size)`	Aloca *size* bytes e retorna o ponteiro para a área de memória alocada. A função também adiciona os *size* bytes de memória para a MV. Em caso de erro, retorna NULL.
`void omp_free(void *ptr)`	Desaloca o espaço de memória apontado por *ptr* e subtrai a memória da MV.

As rotinas *omp_get_max_cpu* e *omp_get_free_cpu* são usadas para obter informações sobre os processadores disponíveis na máquina física que hospeda a VM onde a aplicação está sendo executada. A primeira recebe o número máximo de CPUs disponíveis na máquina física e a última retorna o número de CPUs que podem ser imediatamente alocadas. Estas funções são úteis para determinar o número de *threads* e VCPUs que podem ser utilizadas em uma seção paralela.

Propôs-se também uma primitiva para a elasticidade de memória. A rotina *omp_alloc* é semelhante à função *malloc* fornecida pela *libc*[2], mas também adiciona a quantidade de memória solicitada à MV. Por sua vez, a rotina *omp_free* libera o espaço de memória apontado pelo argumento da função e desaloca a quantidade de memória da MV.

[2] `http://www.gnu.org/software/libc/`

7.3 Avaliação Experimental

Nesta seção, apresentam-se três experimentos usando o OpenMP elástico com o objetivo de demonstrar como a elasticidade pode ser empregada para fornecer funcionalidades adicionais para aplicações OpenMP. No primeiro experimento, a elasticidade é utilizada para melhorar a eficiência de uma aplicação utilizando a quantidade mais apropriada de recursos. No segundo, emprega-se a possibilidade de alocação dinâmica de VCPUs para implementar balanceamento de carga. No terceiro experimento avalia-se o uso de alocação dinâmica de memória. Nestes experimentos, o ambiente computacional utilizado é o mesmo descrito na Seção 6.1.

7.3.1 Experimento 1: Melhorando a eficiência de aplicações

Neste experimento, utiliza-se a elasticidade para melhorar a eficiência de uma aplicação utilizando a quantidade mais apropriada de recursos. A eficiência é a medida de quanto a capacidade de processamento disponível é efetivamente usada. Eficiência próxima da unidade significa que o hardware é efetivamente utilizado, enquanto que baixa eficiência significa que os recursos são desperdiçados [84]. Aumentar o número de processos ou *threads* de uma aplicação paralela, em geral, pode aumentar seu desempenho (*speedup*), mas pode diminuir a sua eficiência. Isso pode ocorrer devido a fatores tais como contenção de memória, comunicação e estrutura do software.

Garantir o uso eficiente de recursos em ambientes de nuvem é bastante importante. Em nuvens públicas é desejável que a eficiência das aplicações seja mantida o mais próximo da unidade, de modo que o desempenho obtido seja linearmente proporcional ao custo da execução. Aplicações pouco eficientes podem resultar em grandes investimentos e ganhos de desempenho pouco significativos. No caso de nuvens privadas, o uso eficiente dos recursos permite que mais usuários/aplicações possam compartilhar a infraestrutura subjacente.

Encontrar a quantidade de recursos mais eficiente não é uma tarefa fácil, considerando que aplicações podem apresentar resultados diferentes, dependendo dos dados de entrada e parâmetros utilizados. Assim, é possível empregar a elasticidade fornecida pelo OpenMP elástico para permitir que a aplicação encontre automaticamente os recursos mais adequados para manter a sua eficiência em um determinado nível.

Neste teste, utilizou-se uma implementação do problema de transferência de calor, que consiste na resolução de uma equação diferencial parcial em cada iteração para determinar a variação da temperatura dentro de um domínio 2D quadrado. A aplicação tem algumas partes sequenciais que afetam a sua escalabilidade, assim, há um limite para o número de VCPUs que pode ser usado de forma eficiente. A aplicação foi testada com domínios de 8192×8.192 e 16.384×16.384 células.

A implementação original foi modificada para suportar a nova funcionalidade conforme apresentado no pseudo-código da Figura 7.5. A cada iteração, calcula-se a eficiência obtida com a quantidade atual de *threads*/VCPUs. Se o valor for maior do que um determinado limiar (0,75 neste teste), mais recursos são alocados automaticamente para a próxima iteração. Caso a eficiência permaneça inferior ao valor de referência por duas iterações consecutivas, os últimos recursos alocados são liberados. Todas as alocações e desalocações de recursos são realizadas automaticamente pelo OpenMP.

```
transferência_calor()
    ...
    para cada iteração:
        calcula_eficiencia()

        se eficiência >= 0.75
            threads++ //OpenMP automaticamente adiciona VCPU

        se eficiência < 0.75 por duas iterações consecutivas
            threads-- //OpenMP automaticamente remove VCPU

        calcula_calor_OpenMP(threads)
```

Figura 7.5: Pseudo-código da aplicação OpenMP com alocação de VCPUs de acordo com a eficiência obtida.

Comparou-se a implementação usando OpenMP elástico com um mecanismo baseado em monitoramento de máquina virtual. Considerando a falta de mecanismos com essa característica para aplicações de memória compartilhada, foi necessário desenvolver uma solução que emprega elasticidade vertical para permitir a alocação de novas VCPUs. O mecanismo proposto se assemelha aos fornecidos por provedores de nuvem pública, os quais baseiam-se na coleta de informações da máquina virtual e no uso de regras para disparar ações de elasticidade.

Neste mecanismo, um controlador de elasticidade monitora a MV e se todas as VCPUs alocadas apresentarem uso superior a 90%, uma nova VCPU é adicionada. As verificações são feitas a cada 5 segundos. Neste caso, a alocação de VCPUs é feita pelo controlador e as novas VCPUs são percebidas pela aplicação OpenMP de modo automático[3].

Os resultados obtidos com ambas as abordagens são apresentadas nas Figuras 7.6 e 7.7. No início da execução, as MVs foram configuradas com uma única VCPU e 20 GB RAM. Para cada abordagem, executou-se 50 iterações da aplicação. Os testes foram repetidos 20 vezes para garantir a validade dos experimentos (90% de confiança).

A Figura 7.6-a mostra o número de VCPUs atribuídas ao longo da execução para a aplicação implementada com OpenMP elástico. Note que o número de VCPUs é incrementado enquanto a eficiência se mantém acima de 0,75. Na Figura 7.6-b a eficiência alcançada em cada iteração é apresentada. Como pode-se observar, após o período de adaptação (iteração 1 a 20), a eficiência média alcançada é de 0,78. Note-se que os resultados são diferentes para os diferentes tamanhos de dados de entrada, apresentando uma escalabilidade melhor para o domínio de tamanho maior. Isso demonstra que o uso da elasticidade em nível de programação pode considerar particularidades de cada execução (tamanho do domínio, neste caso) para alcançar melhores resultados.

Por sua vez, os resultados apresentados na Figura 7.7 mostram que a solução baseada no monitoramento da MV não é capaz de garantir a eficiência exigida. Isto ocorre devido à incapacidade do mecanismo para obter informações sobre a aplicação, o que é necessário para decidir se os recursos devem ser alocados ou não.

É possível observar que a abordagem baseada no monitoramento é mais agressiva, e após 16 iterações todas as 20 VCPUs disponíveis são alocadas para a aplicação, independentemente do tamanho da entrada. Percebe-se, porém, que o número de *threads* utilizado é sempre inferior ao número de VCPUs alocadas. Isso se deve ao fato de que a aplicação detecta as novas VCPUs uma iteração após elas terem sido alocadas, afetando a eficiência da aplicação (abaixo de 0,7, em média). Este resultado mostra que para proporcionar elasticidade para aplicações paralelas é necessário mais do que informações sobre o uso de recursos da MV. É também necessário que a

[3]Por padrão, o número de *threads* de uma região paralela OpenMP é definida pela quantidade de CPUs/cores da máquina subjacente

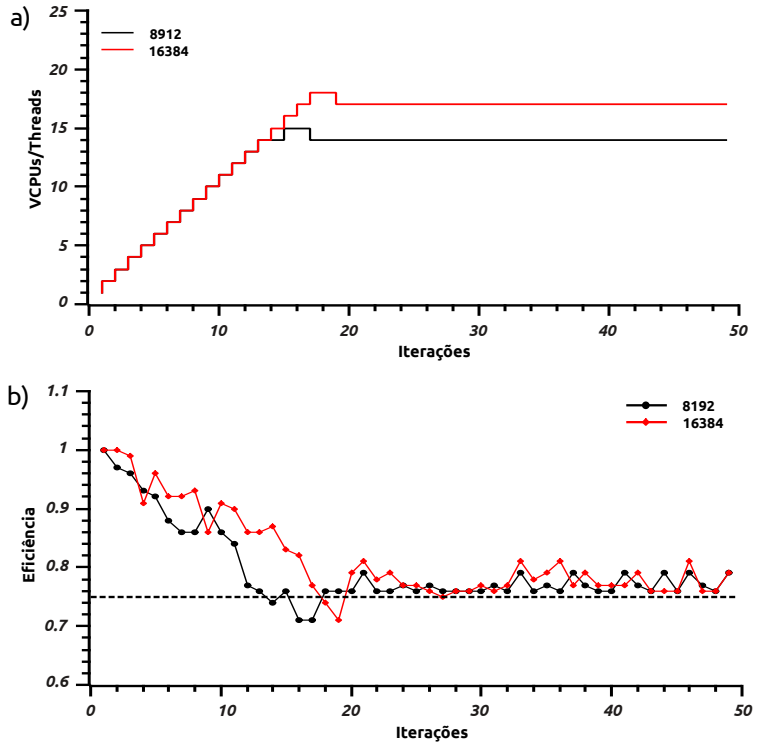

Figura 7.6: OpenMP elástico: a) Número de VCPUs e *threads* utilizados. b) Eficiência.

aplicação esteja preparada para a alocação dinâmica dos recursos e o mecanismo de elasticidade considere as particularidades de cada aplicação.

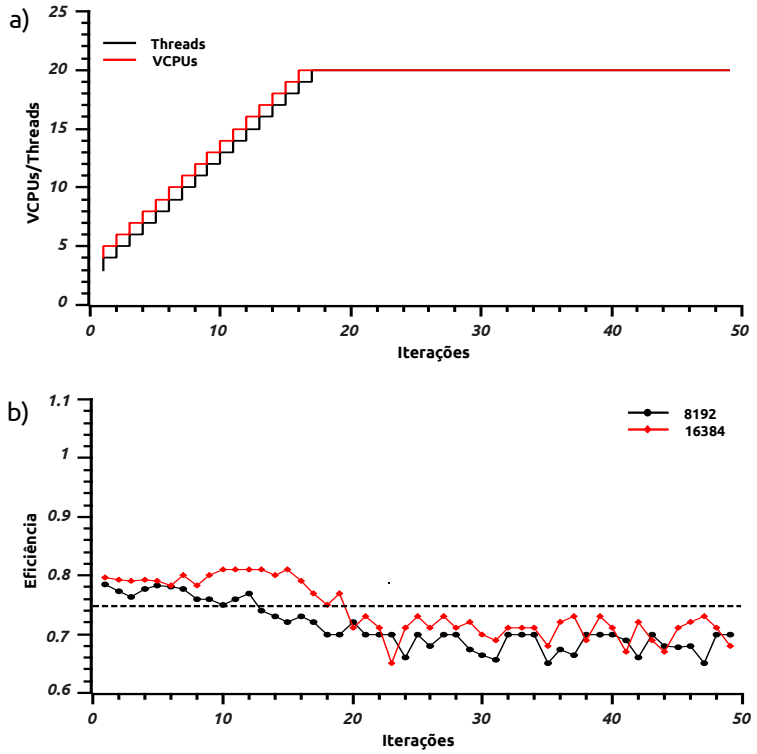

Figura 7.7: Mecanismo baseado em monitoramento: a) Número de VCPUs e *threads* utilizados. b) Eficiência.

7.3.2 Experimento 2: Balanceamento de Carga

O objetivo deste experimento é avaliar o uso de OpenMP elástico na construção de mecanismos de balanceamento de carga em aplicações distribuídas que utilizam OpenMP em sua implementação. Para descobrir o potencial desta abordagem, desenvolveu-se uma aplicação sintética com as características necessárias para a execução dos testes.

A aplicação é composta por P processos que executam N iterações de uma determinada computação (conjunto de operações sobre vetores) e ao final de cada iteração há uma sincronização entre os processos, como apresentado na Figura 7.8. A fase de computação é paralelizada com OpenMP. Para forçar o desbalanceamento, as cargas de trabalho (tamanho dos vetores) são distribuídas de acordo com a fórmula

$$carga\_trabalho = R \times 200.000$$

na qual R é um valor aleatório (entre 2 e 6 nesse teste). Assim, o tempo de processamento é maior em alguns processos do que em outros.

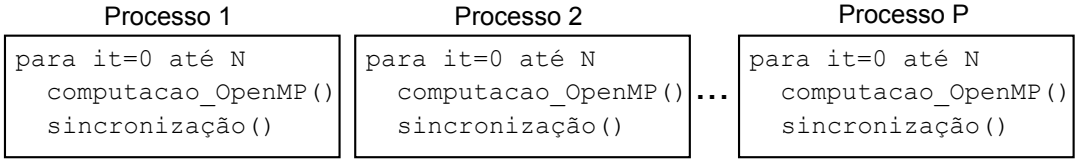

Figura 7.8: Aplicação sintética para avaliação de balanceamento de carga.

Considerando que existe uma operação de sincronização, os processos que possuem carga de trabalho menor tendem a ficar ociosos enquanto aguardam pelo término da computação dos processos com carga superior. Como consequência, o tempo de execução de iteração é determinada pelo tempo do processo mais lento.

A Figura 7.9 apresenta os resultados da aplicação implementada com OpenMP original. Como ambiente de execução foram empregadas duas MVs configuradas com duas VCPUs e 20 GB RAM. Foram realizadas 10 execuções da aplicação, no entanto, considerando a aleatoriedade presente na aplicação, a Figura 7.9 apresenta os resultados em um caso médio.

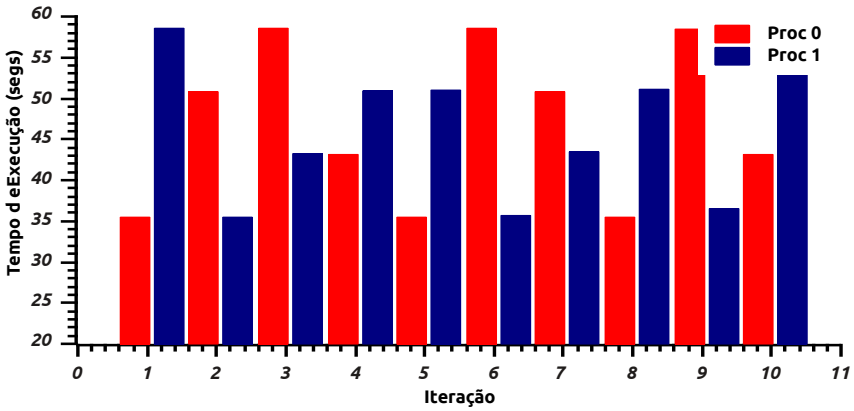

Figura 7.9: Tempo de execução obtidos com cargas desbalanceadas (OpenMP).

É possível observar o desbalanceamento de carga na execução de 10 iterações, com diferenças de até 60% entre o tempo de execução apresentados para ambos os processos. O tempo total obtido neste experimento foi de 527 segundos.

O balanceamento de carga foi implementado por meio da alocação de VCPUs/*threads* para cada processo de acordo com a carga de trabalho atribuída a cada iteração. Dessa forma, espera-se que os tempos de execução de todos os processos seja semelhantes (considerando a relação carga/VCPU) e, dessa forma, reduza-se o tempo total de execução da aplicação. Além disso, a elasticidade permite que os recursos físicos disponíveis na máquina hospedeira possam ser atribuídos dinamicamente aos processos de modo a reduzir ainda mais o tempo de execução. A Figura 7.10 mostra os resultados da aplicação utilizando o OpenMP elástico (caso médio de 10 execuções).

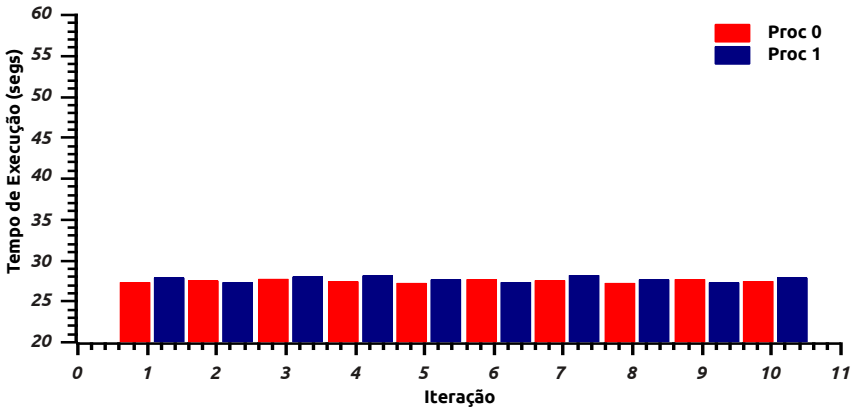

Figura 7.10: Tempo de execução obtidos com cargas balanceadas (OpenMP elástico).

Conforme pode-se observar, o balanceamento de carga implementado funcionou conforme o esperado, apresentando diferenças no tempo de execução da iteração de no máximo 8%, nesta execução. O uso combinado de alocação dinâmica de recursos e de balanceamento de carga reduziu o tempo total de execução para 270 segundos.

7.3.3 Experimento 3: Alocação dinâmica de memória

A capacidade de modificar dinamicamente a memória de uma MV em tempo de execução representa um recurso importante para aplicações com requisitos dinâmicos de memória. Este recurso permite manter memória suficiente para alocar os dados da aplicação na memória residente[4] e, consequentemente, preservar o desempenho da aplicação. Além disso, a memória não utilizada pode ser liberada quando não for mais demandada pela aplicação.

Nesta seção, apresenta-se um experimento usando as rotinas fornecidas pelo OpenMP elástico e compara-se seus resultados com os resultados obtidos com o mecanismo de solução baseada em monitoramento proposto no trabalho de Moltó et al. [102]. Empregou-se a mesma aplicação sintética apresentado na Seção 7.3.2 com dois cenários distintos: (1) carga de trabalho crescente e (2) de carga de trabalho oscilante. No primeiro, a carga de trabalho é aumentada a cada duas iterações, e no segundo, as demandas por memória alternam entre altas e baixas a cada duas iterações. Em ambos os testes, foram empregadas duas MVs com 2 VCPUs e, inicialmente, 2 GB RAM. Implementou-se um programa auxiliar para coletar informações sobre a memória total alocada para a MV e a memória residente utilizada pela aplicação. Tais informações são obtidas por meio dos comandos *free* e *ps* disponíveis no sistema operacional GNU/Linux e são coletadas a cada 2 segundos.

Primeiramente, avaliou-se o uso das rotinas de alocação dinâmica de memória fornecidas pelo OpenMP elástico. A aplicação foi modificada através da substituição das rotinas *malloc*'s e *free*'s pelas rotinas *omp_alloc* e *omp_free*. A Figura 7.11 mostra os resultados usando esta abordagem. Em ambos os cenários, as rotinas de elasticidade trabalharam de forma eficiente na alocação e desalocação de recursos, fornecendo memória suficiente para executar a aplicação. Também é possível observar que toda a aplicação foi mantida na memória residente durante

[4] A memória residente é a parte da memória de um processo que está alocada efetivamente na RAM.

todo o período de execução. A diferença entre a memória alocada (linha vermelha) e a memória utilizada pela aplicação (linha preta) é bem próximo dos 2 GB inicialmente atribuídos à VM. Com esta abordagem, o tempo total de execução foi de aproximadamente 267 segundos para o primeiro cenário e 278 segundos para o segundo.

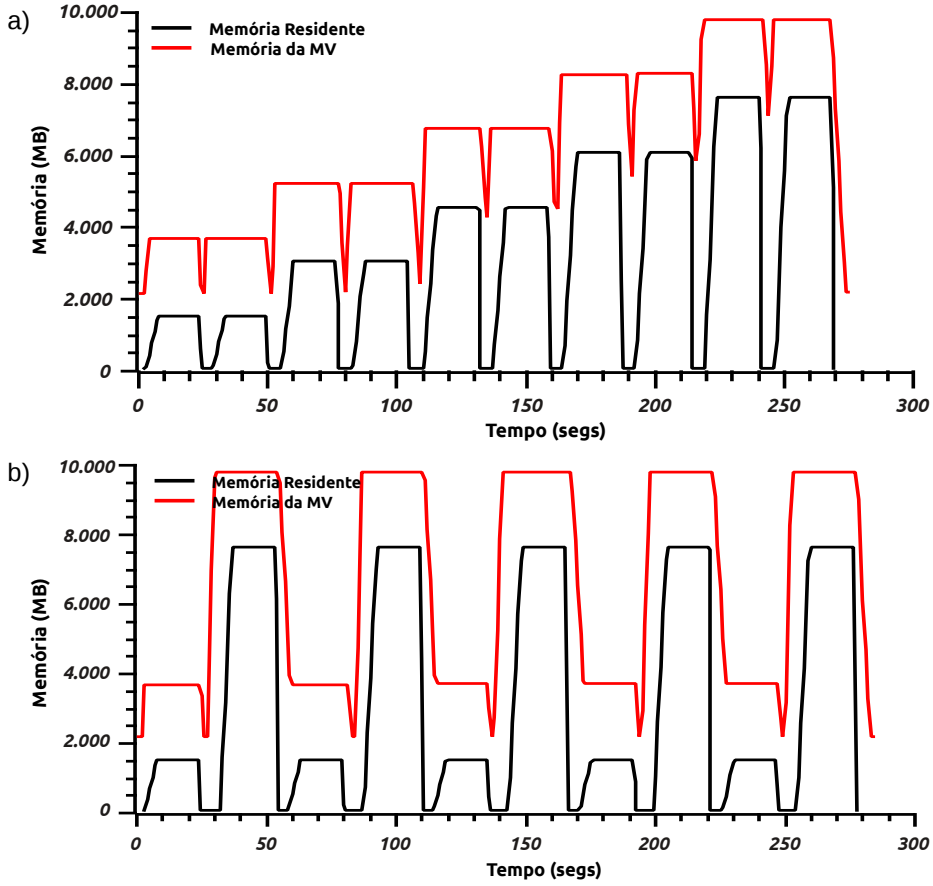

Figura 7.11: Alocação de memória usando OpenMP elástico: a) cargas crescente. b) carga oscilante.

No segundo teste, utilizou-se o sistema baseado no monitoramento das MVs proposto por Moltó et al. [102]. Neste caso, nenhuma modificação na aplicação foi necessária. O mecanismo utiliza a expressão

$$\textit{Memória da MV} = \textit{Memória utilizada} \times (1+\textit{PPA})$$

para calcular quantidade de memória a ser alocada, na qual PPA significa o *Percentual de provisionamento adicional de memória*[5] que indica a quantidade de memória a ser alocada ou

[5] *Memory Overprovisioning Percentage* (MOP), no artigo original.

desalocada. O mecanismo foi configurado para coletar informações sobre o uso de memória a cada 1 segundo e PPA=50%.

Os resultados obtidos com este mecanismo são apresentados na Figura 7.12. No primeiro cenário, o mecanismo obteve êxito para proporcionar a memória necessária para a aplicação, mesmo na maioria das vezes não mantendo toda a aplicação na memória residente, o que afetou o desempenho da aplicação. O tempo total de execução neste cenário foi de aproximadamente 322 segundos, 55 segundos mais lento que na abordagem utilizando o OpenMP elástico. No segundo cenário, o mecanismo falhou ao alocar memória para o pico de carga de trabalho e a aplicação foi abortada no início da segunda iteração.

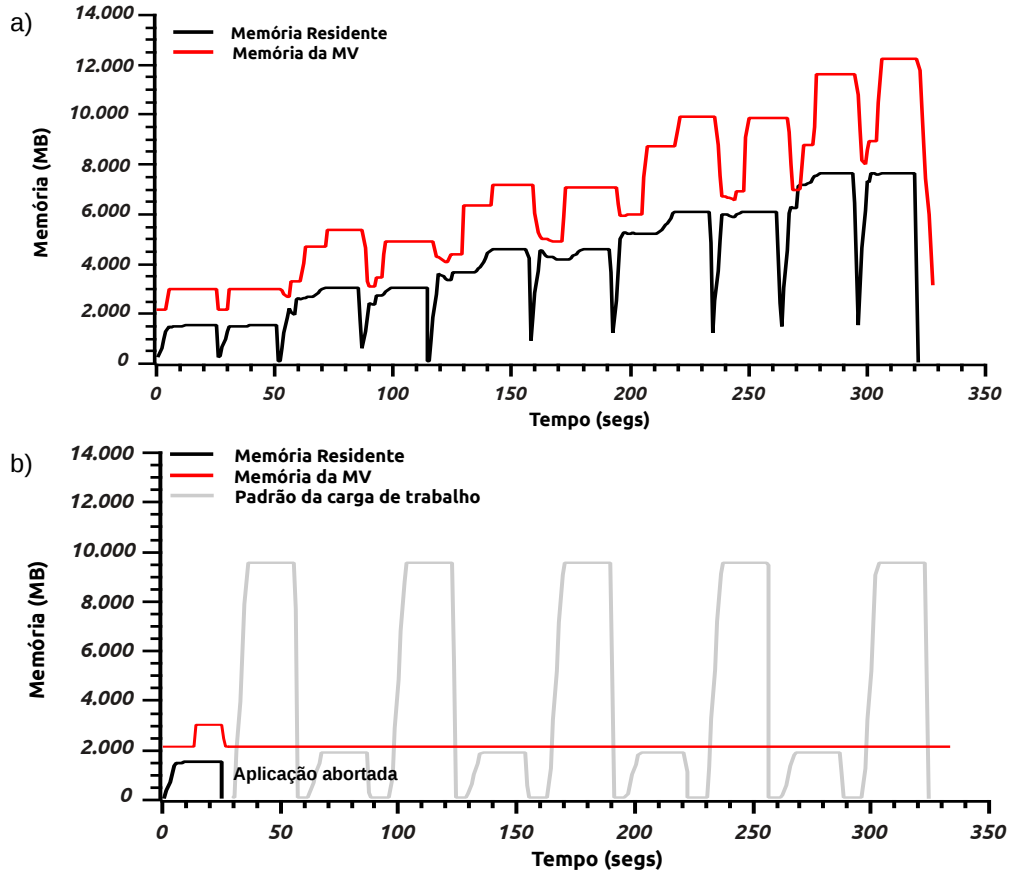

Figura 7.12: Alocação de memória usando mecanismo proposto por Moltó et al.: a) cargas crescente. b) carga oscilante.

7.4 Considerações finais

Este capítulo descreve um mecanismo implementado utilizando o *framework* Cloudine para fornecer elasticidade de modo automático para aplicações OpenMP. Neste mecanismo a API proposta estende o padrão OpenMP original adicionando modificações para permitir o provisionamento dinâmico de recursos. As extensões de elasticidade são baseadas em adaptações das diretivas OpenMP e na adição de um conjunto de rotinas de nível de usuário para a API OpenMP.

Os resultados mostram que OpenMP elástico foi usado com sucesso para adicionar novas características às aplicações, sendo empregado para melhorar a eficiência da aplicação, para implementar o balanceamento de carga e incluir alocação dinâmica de memória. Além disso, o uso de OpenMP para a construção de aplicações elásticas apresentou resultados superiores quando comparados aos resultados dos sistemas baseados em monitoramento. Esta vantagem se deve ao fato de que o OpenMP elástico permite que os programadores construam controladores de elasticidade apropriados para cada aplicação ou cenário, com a vantagem de não se preocuparem explicitamente com a alocação dinâmica de recursos, que é tratada de modo transparente pela API.

CAPÍTULO 8

CONCLUSÃO

Este trabalho teve como objetivo apresentar uma abordagem adequada para a exploração da elasticidade em aplicações científicas, sob a motivação de que que os mecanismos presentes no estado-da-arte apresentam uma série de limitações ao fornecer suporte à alocação dinâmica de recursos para este tipo de aplicação.

Na proposta apresentada, a elasticidade é controlada em nível de programação, o que significa que todo o controle da elasticidade é incorporado ao código-fonte e passa a fazer parte da lógica da aplicação. Isso permite que a estrutura interna e o comportamento de execução possam ser considerados na construção de soluções elásticas especializadas para cada aplicação. Além disso, as aplicações ganham a capacidade de gerenciar a alocação de seus próprios recursos de modo automático, sem a necessidade de mecanismos externos ou da interação do usuário.

Essa possibilidade de considerar a alocação de recursos como parte da lógica do programa dá origem a um novo paradigma para o projeto e o desenvolvimento de aplicações. Neste paradigma, os recursos passam a ser tratados como elementos variáveis de um programa, podendo ser instanciados e modificados ao longo da execução. Essa característica torna possível o desenvolvimento de aplicações elásticas capazes de adaptar o seu próprio ambiente de execução de acordo com suas demandas ou de modo a adaptá-lo a um cenário específico, relacionado a fatores como custo e desempenho.

As vantagens da abordagem proposta são demonstradas nos experimentos apresentados neste trabalho (Capítulos 6 e 7). A primeira delas, é a possibilidade de implementar aplicações com funcionalidades que não poderiam ser oferecidas caso mecanismos de elasticidade tradicionais tivessem sido empregados. Em geral, isso só é possível porque o controle de elasticidade incorporado à aplicação permite que as características de implementação e de execução das aplicações sejam usadas no desenvolvimento de soluções sob medida para cada situação.

Isso fica claro nos experimentos realizados com o modelo climático OLAM, nos quais empregou-se a elasticidade para adicionar novas funcionalidades ao modelo: alocação dinâmica de VCPUs e memória, e balanceamento de carga. No OLAM, a alocação dinâmica de VCPUs é utilizada para fornecer os recursos de processamento apropriados para cada fase da aplicação. Utilizando um mecanismo tradicional, não seria possível implementar essa funcionalidade, uma vez que estes não são capazes de considerar o comportamento da execução para determinar quando alocar ou liberar recursos.

De modo similar, a solução para alocação dinâmica de memória implementada diretamente no código-fonte do modelo apresentou resultados bastante superiores quando comparados àqueles obtidos com o mecanismos externo proposto por Moltó et al. [102]. Enquanto o primeiro consegue alocar memória suficiente para a aplicação de modo satisfatório, o uso do segundo piora o desempenho da aplicação e, em alguns casos, causa o término da aplicação por falta de memória.

A importância de se considerar as particularidades da aplicação na elaboração de soluções de elasticidade também pode ser observada na implementação de balanceamento de carga no OLAM. Ao se considerar as cargas de trabalho (a parte da malha) atribuídas a cada processo, foi possível implementar o balanceamento de carga por meio da redistribuição dos recursos de modo que a relação carga/recurso fosse equivalente em cada um deles. Os resultados mostram que o uso do balanceador proposto melhorou o tempo de execução do modelo em cerca de 7%, usando a mesma quantidade de recursos. O balanceamento de carga como proposto neste experimento não foi encontrado na literatura.

Outra vantagem da abordagem proposta é a possibilidade de combinar as funcionalidades das primitivas de elasticidade com as oferecidas por outras plataformas de elasticidade. Um exemplo de combinação de *frameworks* é apresentado nos experimentos com a aplicação de montagem de genomas SAND implementada com *framework Work Queue* no modelo mestre-escravo. As primitivas de elasticidade foram utilizadas para permitir que a aplicação alocasse automaticamente máquinas virtuais para hospedar novos processos escravos, eliminando a necessidade do uso de escalonadores prevista pelo *Work Queue*.

As vantagens da exploração da elasticidade em nível de programação são se limitam apenas às aplicações, mas estendem-se também para as bibliotecas de programação paralela. Neste trabalho, apresentou-se uma versão com suporte à elasticidade da biblioteca OpenMP do GCC (*libgomp*). A biblioteca original do OpenMP foi estendida de modo que as ações de elasticidade sejam executadas automaticamente por suas diretivas de compilação (*pragmas*). Dessa forma, a elasticidade continua sendo oferecida em nível de programação (e garantindo seus benefícios) de modo mais transparente, no qual o programador não precisa se preocupar tanto com a lógica envolvida na alocação dos recursos.

Ao se analisar os resultados obtidos neste trabalho, pode-se afirmar que a abordagem proposta é, de fato, adequada para fornecer elasticidade para aplicações científicas, uma vez que consegue-se superar todas as limitações apresentadas pelas soluções presentes no estado-da-arte: (1) as aplicações passam a serem capazes não só de usar recursos adicionais, mas também de requisitá-los quando necessário; (2) o controle da elasticidade considera os elementos internos da aplicação, assim como as informações provenientes das máquinas virtuais e da nuvem; (3) as primitivas de elasticidade permitem a exploração da elasticidade horizontal e vertical, tornando possível a alocação desde uma única VCPU até um cluster virtual completo; (4) a abordagem não é restrita a nenhum modelo de programação específico, podendo-se construir soluções sob medida para cada aplicação. Considerando o exposto, pode-se concluir que os objetivos do trabalho foram atingidos de forma satisfatória.

Trabalhos futuros incluem o aperfeiçoamento do *framework* Cloudine de modo a oferecer suporte a outras nuvens, incluir novas primitivas na API de elasticidade e melhorar seu desempenho. Outro trabalho futuro é a adaptação de uma biblioteca do padrão MPI-2 de modo a combinar a criação dinâmica de processos e a alocação elástica de recursos.

Finalmente, uma nova perspectiva de trabalho é o desenvolvimento de uma abordagem para a exploração da elasticidade baseada em anotações (ou diretivas) inseridas no código-fonte da aplicação. Cada anotação corresponde a um procedimento de alocação de recursos que é implementado externamente (como um *script*, por exemplo). Dessa forma, é pode-se alterar o comportamento elástico da aplicação sem a necessidade de modificar seu código-fonte ou recompilá-la.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Amazon Web Services. Disponível em: http://aws.amazon.com/. Acessado em Mar./2014.

[2] CCIF - The Cloud Computing Interoperability Forum. Disponível em: http://www.cloudforum.org/. Acessado em Mar./2012.

[3] CERNVM. Disponível em: http://cernvm.cern.ch/portal/. Acessado em Mar./2014.

[4] CloudSigma. Disponível em: http://www.cloudsigma.com/. Acessado em Mar./2014.

[5] Condor - High Throughput Computing. Disponível em: http://research.cs.wisc.edu/htcondor/. Acessado em Mar./2014.

[6] Facebook. Disponível em: http://www.facebook.com/. Acessado em Mar./2014.

[7] Flick. Disponível em: http://www.flickr.com/. Acessado em Mar./2014.

[8] GoGrid. Disponível em: http://www.gogrid.com/. Acessado em Mar./2014.

[9] Google. Disponível em: http://www.google.com/. Acessado em Mar./2014.

[10] Google App Engine. Disponível em: https://appengine.google.com/. Acessado em Mar./2014.

[11] Microsoft Azure. Disponível em: http://www.windowsazure.com/. Acessado em Mar./2014.

[12] NASA Nebula Cloud Computing Platform. Disponível em: http://www.nasa.gov/open/plan/nebula.html. Acessado em Mar./2014.

[13] OpenStack. Disponível em: http://openstack.org/. Acessado em Mar./2014.

[14] Profitbricks. Disponível em: https://www.profitbricks.com/. Acessado em Mar./2014.

[15] Rackspace. Disponível em: http://www.rackspace.com/. Acessado em Mar./2014.

[16] RightScale. Disponível em: http://www.rightscale.com/. Acessado em Mar./2014.

[17] Sabalcore. Disponível em: http://www.sabalcore.com/. Acessado em Mar./2014.

[18] Salesforce. Disponível em: http://www.salesforce.com/. Acessado em Mar./2014.

[19] Scalr. Disponível em: http://scalr.net/. Acessado em Mar./2014.

[20] Twitter. Disponível em: https://www.twitter.com/. Acessado em Mar./2014.

[21] VMWare vSphere. Disponível em: http://www.vmware.com/products/vsphere/. Acessado em Mar./2014.

[22] Xen Credit Scheduler. Disponível em: `http://wiki.xenproject.org/wiki/Credit_Scheduler/`. Acessado em Mar./2014.

[23] Youtube. Disponível em: `http://www.youtube.com/`. Acessado em Mar./2014.

[24] D. Agrawal, A. El Abbadi, S. Das, e A. J. Elmore. Database scalability, elasticity, and autonomy in the cloud. *Proceedings of the 16th International conference on Database systems for advanced applications - Volume Part I*, DASFAA'11, páginas 2–15. Springer, 2011.

[25] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, e M. Zaharia. A view of cloud computing. *Communications of the ACM*, 53:50–58, 2010.

[26] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph., R. H. Katz., A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, e M. Zaharia. Above the clouds: A berkeley view of cloud computing. Relatório Técnico UCB/EECS-2009-28, EECS Department, University of California, Berkeley, 2009.

[27] F. M. Aymerich, G. Fenu, e S. Surcis. An approach to a Cloud Computing network. *Proceedings of the 1st Applications of Digital Information and Web Technologies*, ICADIWT'08, páginas 113–118, 2008.

[28] L. Badger, R. Patt-Corner, e J. Voas. Draft cloud computing synopsis and recommendations recommendations of the national institute of standards and technology. *Nist Special Publication*, 146:84, 2011.

[29] A. O. Ben-Yehuda, M. Ben-Yehuda, A. Schuster, e D. Tsafrir. The resource-as-a-service (raas) cloud. *Proceedings of the 4th USENIX conference on Hot Topics in Cloud Computing*, HotCloud'12, páginas 12–12. USENIX, 2012.

[30] T. Bicer, D. Chiu, e G. Agrawal. A framework for data-intensive computing with cloud bursting. *Proceedings of the 2011 Companion on High Performance Computing Networking, Storage and Analysis Companion*, SC'11, páginas 5–6. ACM, 2011.

[31] P. Bientinesi, R. Iakymchuk, e J. Napper. HPC on competitive cloud resources. B. Furht e A. Escalante, editors, *Handbook of Cloud Computing*, páginas 493–516. Springer, 2010.

[32] M. L. Bote-Lorenzo, Y. A. Dimitriadis, e E. Gómez-Sánchez. Grid characteristics and uses: A grid definition. F. F. Rivera, M. Bubak, A. G. Tato, e R. Doallo, editors, *1st European Across Grids Conference*, volume 2970 of *Lecture Notes in Computer Science*, páginas 291–298. Springer, 2003.

[33] P. Brebner. Is your cloud elastic enough?: performance modelling the elasticity of infrastructure as a service (iaas) cloud applications. *Proceedings of the 3rd joint WOSP/SIPEW International conference on Performance Engineering*, ICPE'12, páginas 263–266. ACM, 2012.

[34] R. Buyya, R. Ranjan, e R. N. Calheiros. Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities. *Proceedings of the International Conference on High Performance Computing and Simulation*, HPC&S'09, páginas 1–11. IEEE, 2009.

[35] R. Buyya, R. Ranjan, e R. N. Calheiros. Intercloud: utility-oriented federation of cloud computing environments for scaling of application services. *Proceedings of the 10th international conference on Algorithms and Architectures for Parallel Processing - Part I*, volume 6081 of *Lecture Notes in Computer Science*, páginas 13–31. Springer, 2010.

[36] R. Buyya, C. Shin Yeo, e S. Venugopal. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. *Proceedings of the 10th International Conference on High Performance Computing and Communications*, HPCC'08, páginas 5–13. IEEE, 2008.

[37] E. Byun, Y. Kee, J. Kim, e S. Maeng. Cost optimized provisioning of elastic resources for application workflows. *Future Generation Computer Systems*, 27(8):1011–1026, outubro de 2011.

[38] A. Calatrava, G. Moltó, e V. Hernandez. Combining grid and cloud resources for hybrid scientific computing executions. *Proceedings of the 3rd International Conference on Cloud Computing Technology and Science*, CloudCom'11, páginas 494–501. IEEE, 2011.

[39] R. N. Calheiros, C. Vecchiola, D. Karunamoorthy, e R. Buyya. The aneka platform and qos-driven resource provisioning for elastic applications on hybrid clouds. *Future Generation Computer Systems*, 28(6):861–870, junho de 2011.

[40] E. Caron, Luis Rodero-Merino, F. Desprez, e A. Muresan. Auto-scaling, load balancing and monitoring in commercial and open-source clouds. Relatório Técnico 7857, INRIA, 2012.

[41] A. Celesti, F. Tusa, M. Villari, e A. Puliafito. Three-phase cross-cloud federation model: The cloud sso authentication. *Proceedings of the 2nd International Conference on Advances in Future Internet*, AFIN'10, páginas 94–101. IEEE, 2010.

[42] R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald, e R. Menon. *Parallel programming in OpenMP*. Morgan Kaufmann Publishers Inc., 2001.

[43] B. Chapman, G. Jost, e R. Pas. *Using OpenMP: Portable Shared Memory Parallel Programming*. The MIT Press, 2007.

[44] C. Chapman, W. Emmerich, F. Galan Marquez, S. Clayman, e A. Galis. Elastic service management in computational clouds. *Proceedings of the International Workshop on Cloud Management*, CloudMan'10, páginas 1–8. IEEE/IFIP, 2010.

[45] T. C. Chieu, A. Mohindra, A. A. Karve, e A. Segal. Dynamic Scaling of Web Applications in a Virtualized Cloud Computing Environment. *Proceedings of the 2009 IEEE International Conference on e-Business Engineering*, ICEBE 2009, páginas 281–286. IEEE, 2009.

[46] N. Chohan, C. Castillo, M. Spreitzer, M. Steinder, A. Tantawi, e C. Krintz. See spot run: using spot instances for mapreduce workflows. *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, HotCloud'10, páginas 1–7. USENIX, 2010.

[47] P. Church e A. Goscinski. Iaas clouds vs. clusters for hpc: A performance study. *Proceedings of the 2nd International Conference on Cloud Computing, GRIDs, and Virtualization*, Cloud Computing'11, páginas 39–45. IARIA, 2011.

[48] R. Costa e F. Brasileiro. On the amplitude of the elasticity offered by public cloud computing providers. Relatório técnico, Federal University of Campina Grande, 2011.

[49] R. Costa, F. Brasileiro, G. Lemos Filho, e D. M. Sousa. escience-as-a-service: Desafios e oportunidades para a criação de nuvens científicas. *Proceedings of the 5th Brazilian e-Science Workshop*, escience'11, páginas 1–8. SBC, 2011.

[50] S. Das. *Scalable and Elastic Transactional Data Stores for Cloud Computing Platforms*. Tese de Doutorado, University of California, Santa Barbara, CA, 2011.

[51] P. De, M. Gupta, M. Soni, e A. Thatte. Caching vm instances for fast vm provisioning: a comparative evaluation. *Proceedings of the 18th international conference on Parallel Processing*, volume 7484 of *Lecture Notes in Computer Science*, páginas 325–336. Springer, 2012.

[52] E. Deelman, G. Singh, M. Livny, B. Berriman, e J. Good. The cost of doing science on the cloud: the montage example. *Proceedings of the 20th ACM/IEEE Conference on Supercomputing*, SC'08, páginas 50:1–50:12. IEEE, 2008.

[53] V. Delgado. Exploring the limits of cloud computing. Dissertação de Mestrado, Kungliga Tekniska Högskolan, Estocolmo, Suécia, 2010.

[54] P. J. Denning. Computing is a natural science. *Communications of the ACM*, 50(7):13–18, julho de 2007.

[55] R. V. Dorneles, R. L. Rizzi, T. A. Diverio, e P. O. A. Navaux. Dynamic Load Balancing in PC Clusters: An Application to a Multi-Physics Model. *Proceedings of the 15th Symposium on Computer Architecture and High Performance Computing*, SBAC-PAD '03, páginas 192–198. IEEE, 2003.

[56] K. Dowd. *High performance computing - RISC architectures, optimization and benchmarks.* O'Reilly & Associates, Inc, 1993.

[57] A. Edlund, M. Koopmans, Z. A. Shah, I. Livenson, F. Orellana, J. Kommeri, M. Tuisku, P. Lehtovuori, K. M. Hansen, H. Neukirchen, e E. Hvannberg. Practical cloud evaluation from a nordic escience user perspective. *Proceedings of the 5th international workshop on Virtualization technologies in distributed computing*, VTDC'11, páginas 29–38. ACM, 2011.

[58] Eucalyptus. Cloud Computing Miths Dispelled. `http://www.eucalyptus.com/learn/what-is-cloud-computing/cloud-myths-dispelled`. Acessado em Mar./2013.

[59] C. Evangelinos e C. N. Hill. Cloud computing for parallel scientific hpc applications: Feasibility of running coupled atmosphere-ocean climate models on amazon's EC2. *Proceedings of the Cloud Computing and Its Applications*, CCA'08, páginas 1–6, 2008.

[60] A. R. Mury F. J. Fernandes, B. Schulze. Neblina – espaços virtuais de trabalho para uso em aplicações científicas. *Anais do XXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, SBRC 2011, páginas 965–972. SBC, 2011.

[61] J. O. Fitó, I. G. Presa, e J. G. Fernández. Sla-driven elastic cloud hosting provider. *Proceedings of the 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, PDP'10, páginas 111–118. Euromicro, 2010.

[62] I. Foster e C. Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., 2003.

[63] G. Fox e D. Gannon. Cloud programming paradigms for technical computing applications. Relatório técnico, Indiana University, Bloomington, IN, 2012.

[64] G. Fox, A. Ho, e E. Chan. Measured characteristics of futuregrid clouds for scalable collaborative sensor-centric grid applications. *Proceedings of the 2011 International Conference on Collaboration Technologies and Systems*, CTS'11, páginas 151–160. IEEE, 2011.

[65] B. Furht. Cloud computing fundamentals. B. Furht e A. Escalante, editors, *Handbook of Cloud Computing*, páginas 3–19. Springer, 2010.

[66] G. Galante e L. C. E. Bona. A survey on cloud computing elasticity. *Proceedings of the International Workshop on Clouds and eScience Applications Management*, CloudAM'12, páginas 263–270. IEEE, 2012.

[67] G. Galante e L. C. E. Bona. Are public clouds elastic enough for scientific computing? *Proceedings of the 3rd International Workshop on Cloud Computing and Scientific Applications*, CCSA 2013, 2013.

[68] G. Galante e L. C. E. Bona. Constructing elastic scientific applications using elasticity primitives. *Proceedings of the 13th International Conference on Computational Science and Its Applications*, volume 7975 of *Lecture Notes in Computer Science*, páginas 281–294. Springer, 2013.

[69] G. Galante e L. C. E. Bona. Supporting elasticity in openmp applications. *Proceedings of the 22nd Euromicro Conference on Parallel, Distributed and Network-based Processing*, PDP'14. Euromicro, 2014.

[70] G. Galante, L. C. E. Bona, P. A. L. Rego, e J. N. Souza. Erha: Execution and resources homogenization architecture. *Proceedings of the 3rd International Conference on Cloud Computing, Grids, and Virtualization*, Cloud Computing'12, páginas 7. IARIA, 2012.

[71] Z. Gong, X. Gu, e J. Wilkes. Press: Predictive elastic resource scaling for cloud systems. *Proceedings of the 6th International Conference on Network and Service Management*, CNSM'10, páginas 9–16. IEEE, 2010.

[72] A. Gupta e D. Milojicic. Evaluation of hpc applications on cloud. Relatório Técnico HPL-2011-132, HP Laboratories, 2011.

[73] Q. He, S. Zhou, D. Kobler, B.and Duffy, e T. McGlynn. Case study for running hpc applications in public clouds. *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC'10, páginas 395–401. ACM, 2010.

[74] Helix Nebula. Helix Nebula Architecture. Relatório técnico, Helix Nebula Consortium, 2012.

[75] J. Heo, X. Zhu, P. Padala, e Z. Wang. Memory overbooking and dynamic control of xen virtual machines in consolidated environments. *Proceedings of the 11th IFIP/IEEE international conference on Symposium on Integrated Network Management*, IM'09, páginas 630–637. IEEE, 2009.

[76] N. R. Herbst, S. Kounev, e R. Reussner. Elasticity in cloud computing: What it is, and what it is not. *Proceedings of the 10th International Conference on Autonomic Computing*, ICAC'13, páginas 23–27. USENIX, 2013.

[77] H. Hiden, P. Watson, e D. Leahy S. Woodman. e-science central: Cloud-based e-science and its application to chemical property modelling. Relatório Técnico CS-TR-1227, School of Comp. Sci. Newcastle University, 2011.

[78] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, e J. Good. On the use of cloud computing for scientific workflows. *Proceedings of the 4th International Conference on eScience*, ESCIENCE'08, páginas 640–645. IEEE, 2008.

[79] IEEE. Cloud Profiles Working Group. Disponível em: `http://standards.ieee.org/develop/project/2301.html/`. Acessado em Mar./2014.

[80] A. Iordache, C. Morin, N. Parlavantzas, e P. Riteau. Resilin: Elastic MapReduce over Multiple Clouds. Rapport de recherche RR-8081, INRIA, outubro de 2012.

[81] A. Iosup, S. Ostermann, N. Yigitbasi, R. Prodan, T. Fahringer, e D. Epema. Performance analysis of cloud computing services for many-tasks scientific computing. *IEEE Transactions on Parallel and Distributed Systems*, 22(6):931–945, 2011.

[82] S. Islam, K. Lee, A. Fekete, e A. Liu. How a consumer can measure elasticity for cloud platforms. Relatório Técnico 680, School of Information Technologies, University of Sydney, 2011.

[83] S. Jha, D. S. Katz, A. Luckow, A. Merzky, e K. Stamou. Understanding scientific applications for cloud environments. R. Buyya, J. Broberg, e A. M. Goscinski, editors, *Cloud Computing: Principles and Paradigms*, capítulo 13, páginas 345–371. John Wiley & Sons, 2011.

[84] A. H. Karp e H. P. Flatt. Measuring parallel processor performance. *Commun. ACM*, 33(5):539–543, maio de 1990.

[85] M. Keller, D. Meister, A. Brinkmann, C. Terboven, e C. Bischof. escience cloud infrastructure. *Proceedings of the 37th EUROMICRO Conference on Software Engineering and Advanced Applications*, SEAA'11, páginas 188–195. IEEE, 2011.

[86] J. O. Kephart e D. M. Chess. The vision of autonomic computing. *Computer*, 36:41–50, 2003.

[87] T. Knauth e C. Fetzer. Scaling non-elastic applications using virtual machines. *Proceedings of the 4th International Conference on Cloud Computing*, CLOUD'11, páginas 468–475. IEEE, 2011.

[88] D. Kondo, B. Javadi, P. Malecot, F. Cappello, e D. P. Anderson. Cost-benefit analysis of cloud computing versus desktop grids. *Proceedings of the 23rd International Symposium on Parallel and Distributed Processing*, IPDPS'09, páginas 1–12. IEEE, 2009.

[89] M. Kupperberg, N. Herbst, J. Kistowski, e R. Reussner. Defining and quantifying elasticity of resources in cloud computing and scalable platforms. Relatório técnico, Karlsruhe Institute of Technology, 2011.

[90] M. A. P. Laureano. *Máquinas Virtuais e Emuladores - Conceitos, Técnicas e Aplicações*. Novatec, 1 edition, 2006.

[91] A. Li, X. Yang, S. Kandula, e M. Zhang. Cloudcmp: comparing public cloud providers. *Proceedings of the 10th Annual Conference on Internet Measurement*, IMC'10, páginas 1–14. ACM, 2010.

[92] J. Li, M. Humphrey, Y. Cheah, Y. Ryu, D. Agarwal, K. Jackson, e C. Ingen. Fault tolerance and scaling in e-science cloud applications: Observations from the continuing development of modisazure. *Proceedings of the 6th International Conference on e-Science*, ESCIENCE'10, páginas 246–253. IEEE, 2010.

[93] J. H. Lienhard e J. H. Lienhard. *A Heat Transfer Textbook - 3rd ed.* Phlogiston Press: Cambridge, Massachusetts, 2008.

[94] H. C. Lim, S. Babu, J. S. Chase, e S. S. Parekh. Automated control in cloud computing: challenges and opportunities. *Proceedings of the 1st Workshop on Automated Control for Datacenters and Clouds*, ACDC'09, páginas 13–18. ACM, 2009.

[95] M. Livny, J. Basney, R. Raman, e T. Tannenbaum. Mechanisms for high throughput computing. *SPEEDUP Journal*, 11(1):1–6, junho de 1997.

[96] W. Lu, J. Jackson, e R. Barga. Azureblast: a case study of developing science applications on the cloud. *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC'10, páginas 413–420. ACM, 2010.

[97] M. Mao e M. Humphrey. A performance study on the vm startup time in the cloud. *Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing*, CLOUD'12, páginas 423–430. IEEE, 2012.

[98] J. Marshall, A. Adcroft, C. Hill, L. Perelman, e C. Heisey. A Finite-Volume Incompressible Navier-Stokes Model for Studies of Ocean on Parallel Computers. *Journal of Geophysical Research*, 102(C3):5753–5766, 1997.

[99] P. Marshall, K. Keahey, e T. Freeman. Elastic site: Using clouds to elastically extend site resources. *Proceedings of the 10th International Conference on Cluster, Cloud and Grid Computing*, CCGRID'10, páginas 43–52. IEEE, 2010.

[100] V. Mauch, M. Kunze, e M. Hillenbrand. High performance cloud computing. *Future Generation Computer Systems*, 29(6):1408–1416, agosto de 2013.

[101] P. Mell e T. Grance. Draft the nist definition of cloud computing. *Nist Special Publication*, 145(6):7, 2009.

[102] G. Moltó, M. Caballer, E. Romero, e C. de Alfonso. Elastic memory management of virtualized infrastructures for applications with dynamic memory requirements. *International Conference on Computational Science, ICCS'13*, volume 18 of *Procedia Computer Science*, páginas 159–168, 2013.

[103] L. R. Moore, K. Bean, e T. Ellahi. Transforming reactive auto-scaling into proactive auto-scaling. *Proceedings of the 3rd International Workshop on Cloud Data and Platforms*, CloudDP'13, páginas 7–12. ACM, 2013.

[104] C. Moretti, A. Thrasher, L. Yu, M. Olson, Scott Emrich, e Douglas Thain. A framework for scalable genome assembly on clusters, clouds, and grids. *IEEE Transactions on Parallel and Distributed Systems*, 23(12):2189–2197, dezembro de 2012.

[105] J. Napper e P. Bientinesi. Can cloud computing reach the top500? *Proceedings of the Workshops on Unconventional High Performance Computing Workshop plus Memory Access Workshop*, UCHPC-MAW'09, páginas 17–20. ACM, 2009.

[106] I. Neamtiu. Elastic executions from inelastic programs. *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS'11, páginas 178–183. ACM, 2011.

[107] P. Neto. Demystifying cloud computing. *Proceedings of the 6th Doctoral Symposium on Informatics Engineering*, DSIE'11, páginas 2971–2978. Faculdade de Engenharia da Universidade do Porto, 2011.

[108] D. Oliveira, F. A. Baião, e M. Mattoso. Migrating Scientific Experiments to the Cloud. HPC in the Cloud. Disponível em: `http://www.hpcinthecloud.com/hpccloud/2011-03-04/migrating_scientific_experiments_to_the_cloud.html` . Acessado em Set./2013.

[109] D. Oliveira e E. Ogasawara. Is cloud computing the solution for brazilian researchers? *International Journal of Computer Applications*, 6(8):19–23, 2010.

[110] P. Ostberg. *Virtual Infrastructures for Computational Science: Software and Architectures for Distributed Job and Resource Management*. Tese de Doutorado, Umea University, Umea, Sweden, 2011.

[111] D. Owens. Securing elasticity in the cloud. *Queue*, 8(5):10:10–10:16, 2010.

[112] D. F. Parkhill. *The challenge of the computer utility*. Addison-Wesley Professional, 1966.

[113] P. Prabhu, T. B. Jablin, Y. Raman, A.and Zhang, J. Huang, H. Kim, N. P. Johnson, F. Liu, S. Ghosh, S. Beard, T. Oh, M. Zoufaly, D. Walker, e D. I. August. A survey of the practice of computational science. *State of the Practice Reports*, SC'11, páginas 19:1–19:12. ACM, 2011.

[114] I. Raicu. *Many-Task Computing: Bridging the Gap between High Throughput Computing and High Performance Computing*. VDM Verlag, 2009.

[115] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, e M. Wilde. Falkon: a fast and light-weight task execution framework. *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, SC'07, páginas 43:1–43:12. ACM, 2007.

[116] D. Rajan, A. Canino, J. A. Izaguirre, e D. Thain. Converting a high performance application to an elastic cloud application. *Proceedings of the 3rd International Conference on Cloud Computing Technology and Science*, CLOUDCOM'11, páginas 383–390. IEEE, 2011.

[117] L. Ramakrishnan, K. R. Jackson, S. Canon, S. Cholia, e J. Shalf. Defining future platform requirements for e-science clouds. *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC'10, páginas 101–106. ACM, 2010.

[118] A. Raveendran, T. Bicer, e G. Agrawal. A framework for elastic execution of existing mpi programs. *Proceedings of the International Symposium on Parallel and Distributed Processing Workshops and PhD Forum*, IPDPSW'11, páginas 940–947. IEEE, 2011.

[119] P. A. L. Rego, E. F. Coutinho, D. G. Gomes, e J. N. Souza. Faircpu: Architecture for allocation of virtual machines using processing features. *Proceedings of 4th International Conference on Utility and Cloud Computing*, UCC'2011, páginas 371 –376. IEEE, 2011.

[120] J. J. Rehr, F. D. Vila, J .P. Gardner, L. S., e M. Prange. Scientific computing in the cloud. *Computing in Science and Engineering*, 12:34–43, 2010.

[121] N. Roy, A. Dubey, e A. Gokhale. Efficient autoscaling in the cloud using predictive models for workload forecasting. *Proceedings of the 4th International Conference on Cloud Computing*, CLOUD'2011, páginas 500–507. IEEE, 2011.

[122] R. Samtaney. Exascale computing: Challenges and opportunities for applied mathematicians and engineers. *Journal of Applied and Computational Mathematics*, 2(1):–, 2012.

[123] D. E. Y. Sarna. *Implementing and Developing Cloud Computing Applications*. CRC Press, 1 edition, 2011.

[124] J. Schad, J. Dittrich, e J. Quiané-Ruiz. Runtime measurements in the cloud: observing, analyzing, and reducing variance. *Proceedings of the Very Large Database Endowment*, 3(1-2):460–471, setembro de 2010.

[125] C. Schepke, N. Maillard, J. Schneider, e H. Heiss. Why Online Dynamic Mesh Refinement is Better for Parallel Climatological Models. *Proceedings of the 2011 23rd International Symposium on Computer Architecture and High Performance Computing*, SBAC-PAD '11, páginas 168–175. IEEE, 2011.

[126] C. Schepke, N. Maillard, J. Schneider, e H. Heiss. Online Mesh Refinement for Parallel Atmospheric Models. *International Journal of Parallel Programming*, 41(4):552–569, 2013.

[127] E. Schimidt. Conversation with Eric Schmidt hosted by Danny Sullivan. Search Engine Strategies Conference 2006. Disponível em: `http://www.google.com/press/podium/ses2006.html`. Acessado em Jan/2012.

[128] P. Sempolinski e D. Thain. A comparison and critique of eucalyptus, opennebula and nimbus. *Proceedings of the 2nd International Conference on Cloud Computing Technology and Science*, CLOUDCOM'2010, páginas 417–426. IEEE, 2010.

[129] U. Sharma, P. Shenoy, S. Sahu, e A. Shaikh. A cost-aware elasticity provisioning system for the cloud. *Proceedings of the 31st International Conference on Distributed Computing Systems*, ICDCS'11, páginas 559–570. IEEE, 2011.

[130] Z. Shen, S. Subbiah, X. Gu, e J. Wilkes. Cloudscale: elastic resource scaling for multi-tenant cloud systems. *Proceedings of the 2nd Symposium on Cloud Computing*, SOCC'11, páginas 5:1–5:14. ACM, 2011.

[131] Y. Simmhan, C. van Ingen, G. Subramanian, e J. Li. Bridging the gap between desktop and the cloud for escience applications. *Proceedings of the 3rd International Conference on Cloud Computing*, CLOUD'10, páginas 474–481. IEEE, 2010.

[132] J. E. Simons e J. Buell. Virtualizing high performance computing. *ACM Operating Systems Review*, 44(4):136–145, dezembro de 2010.

[133] P. Sobeslavsky. Elasticity in cloud computing. Dissertação de Mestrado, Joseph Fourier University, ENSIMAG, Grenoble, France, 2011.

[134] F. R. C. Sousa, L. O. Moreira, e J. C. Machado. Computação em nuvem autônoma: Oportunidades e desafios. *Proceedings of the 1st Workshop on Autonomic Distributed Systems*, WoSiDA'11, páginas 7–10. SBC, 2011.

[135] L. Stout, M. A. Murphy, e S. Goasguen. Kestrel: an xmpp-based framework for many task computing applications. *Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers*, MTAGS'09, páginas 11:1–11:6. ACM, 2009.

[136] C. Tang. Fvd: a high-performance virtual machine image format for cloud. *Proceedings of the 2011 USENIX technical conference*, USENIX'11, páginas 1–18. USENIX Association, 2011.

[137] H. Truong e S. Dustdar. Cloud computing for small research groups in computational science and engineering: current status and outlook. *Computing*, 91(1):75–91, janeiro de 2011.

[138] R. Tudoran, A. Costan, G. Antoniu, e L. Bougé. A performance evaluation of azure and nimbus clouds for scientific applications. *Proceedings of the 2nd International Workshop on Cloud Computing Platforms*, CloudCP'12, páginas 4:1–4:6, 2012.

[139] L. M. Vaquero, L. Rodero-Merino, e R. Buyya. Dynamically scaling applications in the cloud. *ACM Computer Communications Review*, 41:45–52, 2011.

[140] L. M. Vaquero, L. Rodero-Merino, J. Caceres, e M. Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39:50–55, 2008.

[141] N. Vasić, D. Novaković, S. Miučin, D. Kostić, e R. Bianchini. Dejavu: accelerating resource allocation in virtualized environments. *Proceedings of the 17th International conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS'12, páginas 423–436. ACM, 2012.

[142] C. Vecchiola, S. Pandey, e R. Buyya. High-performance cloud computing: A view of scientific applications. *Proceedings of the 2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks*, ISPAN'09, páginas 4–16. IEEE, 2009.

[143] S. Vijayakumar, Q. Zhu, e G. Agrawal. Dynamic resource provisioning for data streaming applications in a cloud environment. *Proceedings of the 2nd International Conference on Cloud Computing Technology and Science*, CLOUDCOM'10, páginas 441–448. IEEE, 2010.

[144] D. Villegas, N. Bobroff, I. Rodero, J. Delgado, Y. Liu, A. Devarakonda, L. Fong, S. Masoud Sadjadi, e M. Parashar. Cloud federation in a layered service model. *Journal of Computer and System Sciences*, 78(5):1330–1344, setembro de 2012.

[145] J. Vöckler, G. Juve, E. Deelman, M. Rynge, e B. Berriman. Experiences using cloud computing for a scientific workflow application. *Proceedings of the 2nd international workshop on Scientific cloud computing*, ScienceCloud'11, páginas 15–24. ACM, 2011.

[146] W. Voorsluys, J. Broberg, e R. Buyya. Introduction to cloud computing. R. Buyya, J. Broberg, e A. M. Goscinski, editors, *Cloud Computing: Principles and Paradigms*, capítulo 1, páginas 3–42. John Wiley & Sons, 2011.

[147] H. Wald. Cloud computing for the federal community. *IA Newsletter*, 13(2), 2010.

[148] R. L. Walko e R. Avissar. The Ocean-Land-Atmosphere Model (OLAM). Part I: Shallow-Water Tests. *Monthly Weather Review*, 136(11):4033–4044, 2008.

[149] L. Wang, J. Tao, M. Kunze, A. C. Castellanos, D. Kramer, e W. Karl. Scientific cloud computing: Early definition and experience. *Proceedings of the 10th International Conference on High Performance Computing and Communications*, HPCC'08, páginas 825–830. IEEE, 2008.

[150] L. Wang, G. von Laszewski, A. J. Younge, X. He, M. Kunze, J. Tao, e C. Fu. Cloud computing: a perspective study. *New Generation Comput.*, 28(2):137–146, 2010.

[151] L. Wang, J. Zhan, W. Shi, e Y. Liang. In cloud, can scientific communities benefit from the economies of scale? *IEEE Transactions on Parallel and Distributed Systems*, 23(2):296–303, fevereiro de 2012.

[152] J. Wilkening, A. Wilke, N. Desai, e F. Meyer. Using clouds for metagenomics: A case study. *Proceedings of the International Conference on Cluster Computing*, Cluster'09, páginas 1–6. IEEE, 2009.

[153] Y. Xiaotao, L. Aili, e Z. Lin. Research of High Performance Computing With Clouds. *Proceedings of the 3rd International Symposium on Computer Science and Computational Technology*, ISCSCT'10, páginas 289–293, 2010.

[154] L. Youseff, M. Butrico, e D. Da Silva. Towards a Unified Ontology of Cloud Computing. *Proceedings of Grid Computing Environments Workshop*, GCE'08, 2008.

[155] Q. Zhang, L. Cheng, e R. Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18, 2010.

[156] J. Zhu, Z. Jiang, e Z. Xiao. Twinkle: A fast resource provisioning mechanism for internet services. *Proceedings of the 30th IEEE Conference on Computer Communications*, INFOCOM'11, páginas 802–810. IEEE, 2011.

APÊNDICE A

CLOUDINE: ARQUITETURA, IMPLEMENTAÇÃO E OPERAÇÃO

O Cloudine (*Cloud Engine*) é o *framework* desenvolvido com o objetivo de oferecer o suporte necessário para o desenvolvimento de aplicações científicas elásticas usando controle em nível de programação e também o de permitir a execução destas aplicações na nuvem de modo transparente para o usuário.

O *framework* oferece ao usuário uma interface para a alocação do ambiente virtual inicial a ser utilizado pela aplicação, uma API que disponibiliza o conjunto de primitivas para a implementação de aplicações elásticas, bem como um ambiente de execução (*runtime environment*) que suporta todas as operações de alocação de desalocação de recursos da nuvem, conforme apresentado na Figura A.1. Detalhes da arquitetura são apresentados na Seção A.1.

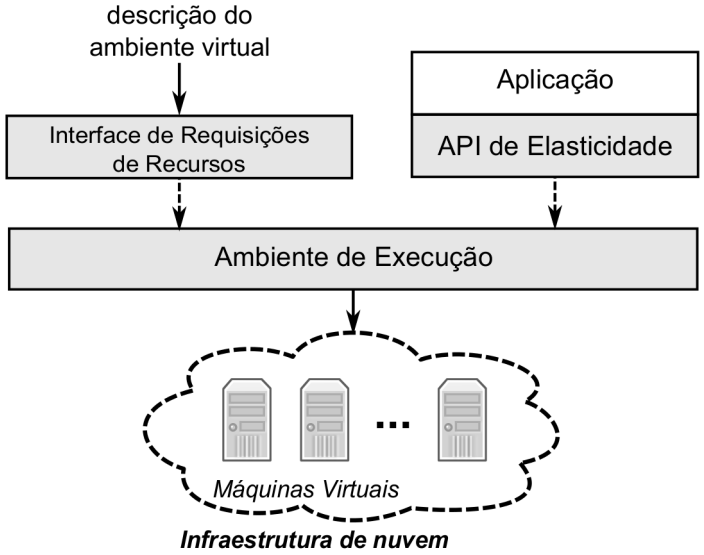

Figura A.1: Arquitetura detalhada do *framework* Cloudine.

A arquitetura do *framework* foi projetada para fornecer flexibilidade na execução de aplicações em nuvens públicas e privadas. Oferecendo suporte a diversas nuvens, as aplicações podem ser executada em qualquer uma delas, sem a necessidade de reimplementação ou alteração do código fonte para garantir portabilidade.

A.1 Arquitetura

Como apresentado anteriormente na Figura A.1, o Cloudine é composto por três componentes principais: (1) o Ambiente de Execução, (2) a Interface de Requisição de Recursos e (3) a API de Elasticidade, descritos em detalhes nas seções a seguir.

A.1.1 Ambiente de Execução

O Ambiente de Execução é o componente que gerencia o provisionamento dinâmico de recursos e realiza toda a interação entre as aplicações elásticas e infraestrutura de nuvem. É por meio deste componente que as solicitações enviadas pela Interface de Requisição de Recursos e pela API de elasticidade são processadas e repassadas à nuvem subjacente. O componente é composto por dois módulos, o Gerenciador de Requisições e o Gerente de Recursos, conforme ilustrado na Figura A.2.

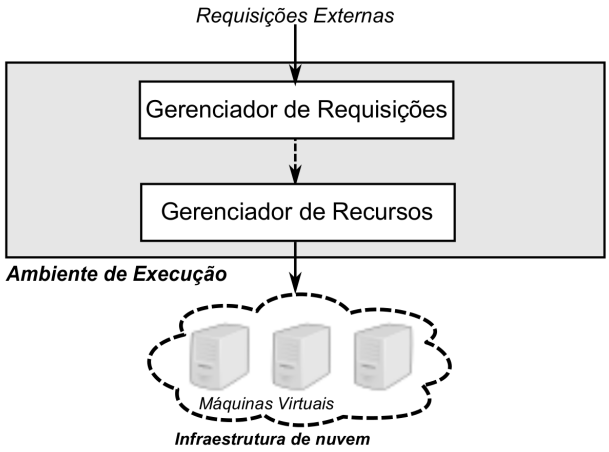

Figura A.2: Módulos do Ambiente de Execução.

O **Gerenciador de Requisições** é o módulo que expõe a interface do Ambiente de Execução. A interface implementa um protocolo de comunicação entre o middleware e demais componentes externos, capaz de suportar as mensagens de requisições por novos ambientes virtuais (originadas na Interface) e mensagens enviadas da API de elasticidade.

O primeiro tipo de mensagem contém os atributos do ambiente a ser criado na nuvem, e incluem número de nós, número de CPUs virtuais, quantidade de memória, imagem de máquina virtual, entre outros. Por sua vez, as requisições da API de elasticidade compreendem um

código de operação e os respectivos argumentos. Ambos os tipos de mensagens são recebidas pelo Gerenciador de Requisições e convertidos em requisições ao Gerenciador de Recursos.

O **Gerenciador de Recursos** é o módulo que gerencia o provisionamento de recursos na infraestrutura de nuvem. O seu papel é permitir o uso de múltiplas nuvens, fornecendo uma camada de abstração para esconder os detalhes das nuvem a serem utilizadas na execução das aplicações. De modo geral, o Gerenciador de Recursos desempenha duas funções: (1) receber os pedidos do Gerenciador de Requisições e alocar ou liberar os recursos na nuvem, e (2) gerenciar as informações sobre os ambientes virtuais.

Dois tipos de requisição são tratados neste módulo: (1) criação de novo ambiente virtual, e (2) as ações de elasticidade da API. No primeiro tipo, o controlador recebe a descrição do ambiente e realiza a alocação de ambiente através de um *Cloud Driver*. Os *Cloud Drivers* são usados para traduzir as requisições em comandos específicos de uma determinada nuvem, usando protocolos como OpenNebula XML-RPC[1], OCCI[2] e EC2[3]. Note que para cada nuvem é necessário implementar um *Cloud Driver* correspondente.

Uma vez criado o ambiente, informações sobre as MVs são recebidos do *VM Daemon* e armazenadas em um banco de dados para uso posterior. O VM Daemon é um módulo automaticamente iniciado na inicialização da VM e seu papel é executar um conjunto de ações internamente na máquina virtual, incluindo a geração de chaves públicas (usadas para acesso remoto), definição do nome da máquina, criação de usuários e execução de *scripts* definidos pelo usuário. Caso não haja recursos físicos disponíveis para a criação das MVs, ou caso algum erro ocorra, o processo é interrompido e mensagens de erro são exibidas pela Interface.

No caso das solicitações originadas pela API de elasticidade, o Gerenciador de Recursos recebe a operação a ser executada (remover VCPU ou adicionar memória, por exemplo), calcula os recursos a serem alocados baseando-se nas informações armazenadas no banco de dados e envia as requisições através do *Cloud Driver* apropriado. Se bem sucedido, o banco de dados é atualizado com a nova configuração do ambiente, caso contrário, um erro é retornado à aplicação.

[1] `http://opennebula.org/documentation:rel4.0:api`
[2] `http://occi-wg.org/`
[3] `http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-query-api.html`

A.1.2 Interface de Requisição de Recursos

O Cloudine fornece uma interface que permite que o usuário solicite o ambiente virtual inicial que será utilizado na execução das aplicações. O ambiente virtual é configurado em um arquivo contendo um conjunto de campos atributo-valor. Os atributos podem incluir o número de nós, número de CPUs virtual (VCPU), quantidade de memória, imagem de máquina virtual ou tipo de instância, configurações de rede, entre outros. Essas configurações são usadas para criar a requisição que é enviada ao Ambiente de Execução, que proporciona a criação de ambiente virtual. Um exemplo de arquivo de configuração é mostrado na Figura A.3.

```
[resources]
env_name=teste
env_type=cluster
env_nodes=4
env_alloc=one
one_ip=192.168.1.102
vcpu=2
memory=512
image=ubuntu.img
network=default
```

Figura A.3: Exemplo de arquivo de configuração de ambiente virtual.

Neste exemplo, solicita-se um cluster de quatro nós, chamado *teste*, a ser alocado em uma nuvem OpenNebula (*one*), cujo endereço de acesso é *192.168.1.102*. Cada nó do cluster possui duas CPUs virtuais, 512 MB de RAM, usa uma imagem chamada *ubuntu.img* e conecta-se a uma rede (*network*) chamada *default*.

Note que os campos do arquivo de configuração pode variar de acordo com a nuvem (ou *driver*) usado, uma vez que os atributos necessários para a configuração das máquinas virtuais são diferentes para cada nuvem.

A.1.3 API de Elasticidade

A API de Elasticidade fornece um conjunto de primitivas que permitem a construção de aplicações elásticas para o Cloudine. Considerando a proposta do *framework*, as primitivas devem permitir a exploração da elasticidade horizontal e vertical. Isso significa que a API deve contemplar primitivas para a alocação e desalocação de máquinas virtuais completas, bem como

dos componentes que a compõem, tais como VCPUs, memória e disco.

Basicamente, a API é formada por uma função de comunicação e um conjunto de funções visíveis para o programador que são utilizadas para solicitar ações de elasticidade e para a aquisição de informações da máquina virtual e da nuvem. A função de comunicação é responsável por enviar mensagens de solicitação para o Ambiente de Execução e por coletar os respectivos códigos de retorno. As demais funções apenas montam as mensagens de requisição e executam a função de comunicação passando a mensagem como argumento.

Essa abordagem permite que novas primitivas possam ser inseridas de modo simples, sem a necessidade de modificações significativas na API. Cada primitiva adicional deve ser inserida na definição do protocolo para que seja reconhecida pelo Gerenciador de Requisições.

É importante destacar que nem todas as nuvens suportam todas as funções oferecidas pela API. Por exemplo, o Amazon EC2 não suporta a alocação dinâmica de memória ou VCPUs, suportando apenas a alocação e desalocação de instâncias de máquinas virtuais. As informações sobre as funções suportadas devem ser implementadas no *Cloud Driver* correspondente.

A.2 Implementação do Protótipo

A implementação do protótipo da arquitetura proposta foi feita utilizando-se as linguagens C, Python e Shell Script e emprega o banco de dados SQLite para o armazenamento das informações dos ambientes virtuais. As comunicações entre os módulos foram implementadas usando sockets TCP/IP.

Até o presente momento, o protótipo oferece suporte para nuvens OpenNebula com virtualização Xen. A escolha por uma nuvem privada OpenNebula foi feita com base na flexibilidade oferecida para a criação de máquinas virtuais. É possível criar cada instância com características próprias de memória e VCPUs, permitindo a configuração de recursos de acordo com as necessidades de aplicação, ao contrário do que ocorre com as plataformas Eucalyptus e OpenStack, nas quais uma classe pré-configurada deve ser escolhida. O Xen Hypervisor foi utilizado devido a capacidade de alocar VCPUs e memória em tempo de execução sem a necessidade de reinicialização da MV, características necessárias para suportar todas as primitivas da API Cloudine.

Considerando que o desenvolvimento do protótipo enfatizou as questões funcionais, foi implementada uma Interface de Requisição de Recursos em linha de comando que disponibiliza apenas as funcionalidades básicas para a requisição dos ambientes virtuais e visualização do estado dos recursos solicitados.

Desenvolveu-se também a API de elasticidade para fornecer o conjunto de primitivas para a construção das aplicações elásticas. Na versão atual, a API oferece suporte às linguagens C e C++ e é constituída por 12 primitivas para a alocação elástica de VCPUs, memória, e máquinas virtuais completas, bem como para coleta de informações sobre o ambiente virtual. A API é disponibilizada por meio da biblioteca compartilhada dinâmica (*dynamic shared library*) `libclne.so`, automaticamente disponibilizada nas máquinas virtuais instanciadas usando o Cloudine. A Tabela A.1 apresenta as primitivas implementadas e suas descrições.

Tabela A.1: Primitivas da API de Elasticidade

Função	**Descrição**
`int clne_add_vcpu(int N)`	Adiciona *N* VCPUs para a MV na qual a aplicação está sendo executada. Retorna 1 em caso de sucesso, 0 caso contrário.
`int clne_rem_vcpu(int N)`	Remove *N* VCPUs da MV na qual a aplicação está sendo executada. Retorna 1 em caso de sucesso, 0 caso contrário.
`int clne_add_node(int N)`	Adiciona *N* nodos ao ambiente virtual (cluster). Retorna 1 em caso de sucesso, 0 caso contrário. Esta função também cria (ou atualiza) um arquivo contendo os nomes e endereços de IP dos nodos que compõem o cluster.
`int clne_rem_node(int N)`	Remove o nodo atual do ambiente virtual (cluster). Retorna 1 em caso de sucesso, 0 caso contrário.
`int clne_add_memory(long int N)`	Adiciona *N* Megabytes de memória para a MV na qual a aplicação está sendo executada. Retorna 1 em caso de sucesso, 0 caso contrário.
`int clne_rem_memory(long int N)`	Remove *N* Megabytes de memória da MV na qual a aplicação está sendo executada. Retorna 1 em caso de sucesso, 0 caso contrário.
`int clne_get_freemem()`	Retorna a quantidade de memória livre da máquina física que hospeda a MV na qual a aplicação está sendo executada.
`int clne_get_maxmem()`	Retorna a quantidade total de memória da máquina física que hospeda a MV.
`int clne_get_mem()`	Retorna a quantidade de memória livre da MV na qual a aplicação está sendo executada.
`int clne_get_freecpu()`	Retorna a quantidade de CPUs livres na máquina física que hospeda a MV.
`int clne_get_maxcpu()`	Retorna a quantidade total de CPUs da máquina física que hospeda a MV.
`int clne_get_vcpus()`	Retorna a quantidade de CPUs da MV na qual a aplicação está sendo executada.

Este conjunto de primitivas foi desenvolvido considerando as plataformas de nuvem OpenNebula. Novas primitivas podem ser adicionadas para acrescentar novas funcionalidades, bem como para suportar outras plataformas de nuvem.

A.3 Operação do *framework*

Para demonstrar como a operação do Cloudine é realizada, apresentam-se nesta seção dois exemplos de uso do *framework*, nos quais descreve-se o processo de alocação de ambientes virtuais e a alocação dinâmica de recursos usando a API elasticidade.

A alocação de um ambiente virtual é ilustrado por meio de um exemplo no qual se instancia um cluster virtual na infraestrutura da nuvem. O processo é composto por nove passos, conforme apresenta a Figura A.4 e descrito na sequência.

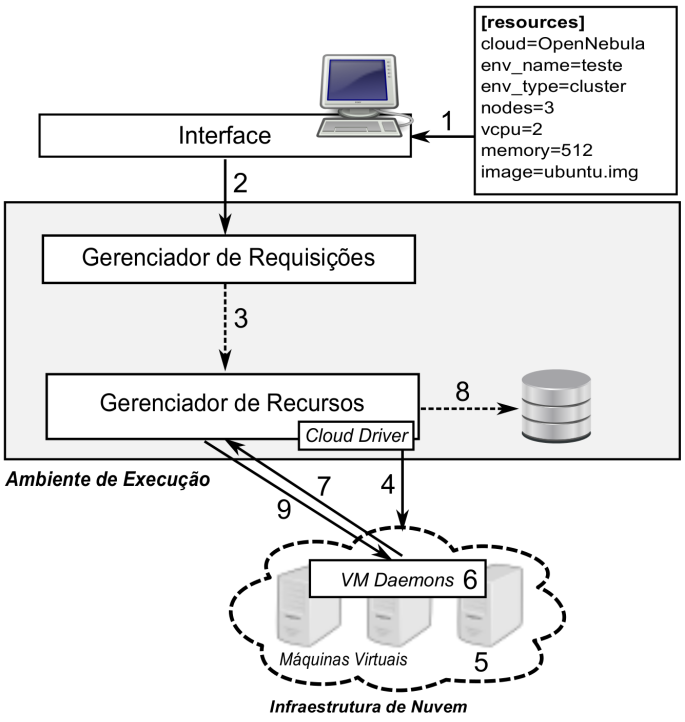

Figura A.4: Alocação de um cluster virtual usando Cloudine

O processo inicia-se com a requisição de um cluster virtual por meio da Interface de Requisição de Recursos (1). Neste exemplo, um cluster de 3 nodos é solicitado em uma nuvem privada OpenNebula. A requisição é então enviada para o Gerenciador de Requisições (2), que recebe a descrição dos recursos, coleta os atributos e os envia para o Controlador de Recursos

(3). Na sequência, o Controlador de Recursos verifica o tipo de requisição, e envia um comando de alocação por meio do *Cloud Driver* apropriado (4). As máquinas virtuais são criadas na nuvem (5) e uma instância do *VM Daemon* é inicializada em cada uma delas (6). Os *VM Daemons* coletam as informações das máquinas e as enviam para o Gerenciador de Recursos (7), que as armazena no banco de dados (8). Particularmente no caso de clusters, o Gerenciador de Recursos envia as chaves públicas para os *VM Daemons*, que as utilizam para configurar o acesso remoto sem o uso de senhas, que é usado em aplicações distribuídas, tais como MPI (9).

Supondo-se que uma aplicação mestre-escravo está sendo executada no cluster virtual requisitado anteriormente e que em um determinado momento o processo mestre solicita a adição de novas VCPUs por meio da primitiva `clne_add_vcpu()`, como ilustra a Figura A.5(1), de modo a processar uma carga de trabalho extra. Quando a primitiva é executada, uma mensagem de requisição é montada, contendo o código da operação de adição de VCPUs, o número de VCPUs a serem adicionadas e a identificação da máquina (2). Na sequência, a mensagem é enviada por meio da função de comunicação (*dispatcher*) para o Ambiente de Execução (3) que envia os comandos para a nuvem (4). Por fim, as duas VCPUs são adicionadas à máquina virtual e um código de retorno é enviado para a aplicação fazendo o caminho reverso (5).

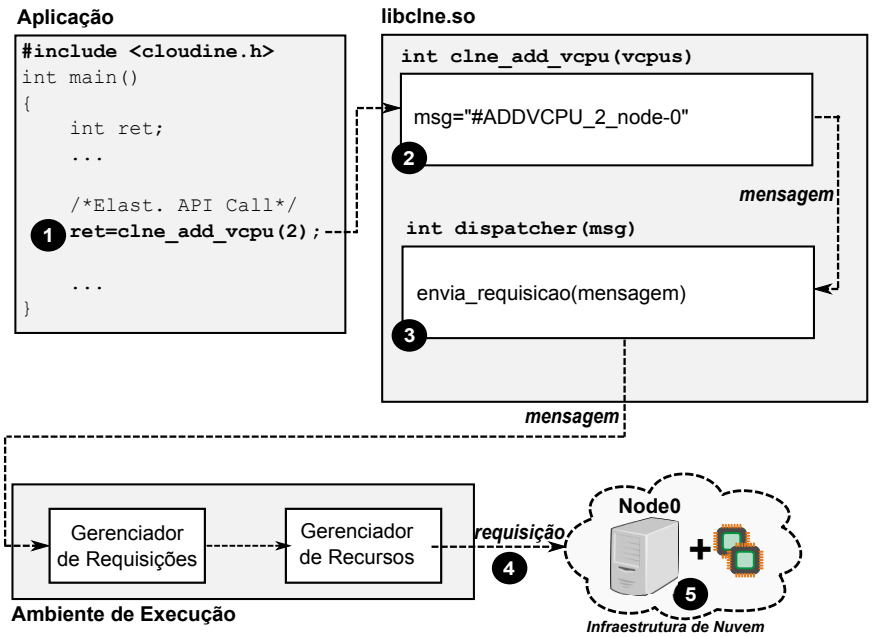

Figura A.5: API de Elasticidade: exemplo de operação

De modo geral, todas as primitivas com os prefixos *clne_add* e *clne_rem* são executadas da mesma forma que a apresentada neste exemplo. As primitivas com prefixo *clne_get* não realizam nenhuma alteração nas máquinas virtuais e ao invés de retornar um código de retorno (sucesso ou erro), retornam informações sobre o ambiente (máquinas virtuais e nuvem).