Cloud Elasticity for HPC Applications: Observing Energy, Performance and Cost

Vinicius Facco Rodrigues, Gustavo Rostirolla, Rodrigo da Rosa Righi,
Cristiano André da Costa, Jorge Luis Victória Barbosa
Applied Computing Graduate Program - Unisinos
Av. Unisinos, 950 – São Leopoldo, RS, Brazil
Email: viniciusfacco@live.com, grostirolla1@gmail.com, {rrrighi,cac,jbarbosa}@unisinos.br

***Abstract*—Elasticity is one of the most known capabilities related to cloud computing, being largely deployed using thresholds. In this way, limits are used to drive resource mangement actions, leading to the following problem statements: How can cloud users set the threshold values to enable elasticity in their cloud applications? And what is the impact of the application's load pattern in the elasticity? This article answers these questions for iterative high performance computing applications, showing the impact of both thresholds and load patterns on application performance and resource consumption. To accomplish this, we developed a reactive and PaaS-based elasticity model called AutoElastic and employed it over a private cloud to execute a numerical integration application. Here, we are presenting an analysis of best practices and possible optimizations regarding the elasticity and HPC pair. Considering the results, we observed that the upper threshold influences the application time more than the lower one.**

***Keywords*—Cloud elasticity, high-performance computing, resource management, self-organizing.**

I. Introdução

O surgimento da computação em nuvem oferece a flexibilidade de gestão dos recursos de uma forma mais dinâmica devido ao fato de ser usada a tecnologia de virtualização para abstrair, encapsular e particionar unidades de processamento. A virtualização possibilita um dos recursos de nuvem mais conhecidos - a elasticidade de recursos [1], [2]. Através do princípio de provisionamento sob demanda, o interesse na capacidade de elasticidade está relacionado aos benefícios que ela pode proporcionar, o que inclui melhorias no desempenho das aplicações, melhor utilização de recursos e redução de custos. Quanto ao desempenho, tem-se a possibilidade de alocação dinâmica de mais recursos na medida que os existentes ficam saturados. A possibilidade de alocar uma pequena quantidade recursos no início da aplicação, evitando provisionamento excessivo, além da necessidade de liberá-los em períodos de carga moderada, enfatiza a justificativa para a redução de custos, o que impacta diretamente na economia de energia e melhor uso de recursos [3].

Alguns esforços recentes focam em explorar a elasticidade em nuvem para serviços Web tradicionais, incluindo um armazenamento de dados transacional, Web Services com intenso tráfego de dados e a execução de aplicações no estilo *bag-of-tasks* [4]. Basicamente, esse cenário cobre organizações que buscam evitar quedas nos serviços com provisionamento fixo para aplicações críticas. A organização típica da elasticidade em tais sistemas utiliza replicação de máquinas virtuais (VMs) e um distribuidor de tarefas centralizado, que também exerce a função de balanceamento de carga. Normalmente, réplicas não se comunicam entre si e a finalização prematura de alguma delas não significa que o sistema ficará indisponível, mas sim em uma nova tentativa de requisição do usuário [5].

Embora a solução mencionada seja empregada com sucesso em aplicações baseadas em um servidor, aplicações científicas de HPC (*High Performance Computing*) fortemente acopladas não podem se beneficiar do uso destes mecanismos. Geralmente, programas desse tipo foram projetados para usar um número fixo de recursos e não podem explorar elasticidade sem um suporte adequado. Em outras palavras, a simples adição de instâncias e a utilização de balanceadores de carga não têm qualquer efeito nessas aplicações uma vez que eles não são capazes de detectar e utilizar esses recursos [6]. Tecnicamente, nos últimos anos, a maioria das aplicações paralelas foram desenvolvidas utilizando o padrão *Message Passing Interface* (MPI) 1.x, que não possui suporte para alterar o número de processos durante a execução da aplicação [7]. No entanto, isso mudou com a versão MPI 2.0, porém esse recurso ainda não é explorado por muitas das implementações MPI disponíveis [4]. Além disso, é necessário um esforço em nível de escrita da aplicação, tanto para alterar manualmente o grupo de comunicadores entre processos quanto para distribuir os dados para usufruir eficazmente de um diferente número de processos. Aliado a isso, o desligamento de uma VM rodando um ou mais processos pode ocasionar a finalização prematura da aplicação, uma vez que os canais de comunicação entre os processos são subitamente encerrados.

Visando proporcionar elasticidade em nuvem para aplicações HPC de uma forma eficiente e transparente, esse artigo apresenta o modelo chamado AutoElastic[1]. A eficiência é abordada considerando-se tempo e utilização de recursos para atender diferentes modificações na carga de trabalho. A transparência, por sua vez, é endereçada pelo provisionamento de elasticidade em nível de middleware. Sendo AutoElastic um gerente baseado em PaaS (*Platform as a Service*), o programador não precisa alterar qualquer linha de código-fonte do aplicativo para tirar proveito dos novos recursos. O modelo proposto assume que a aplicação HPC alvo é iterativa, isto é, tem um laço de execução para distribuição de tarefas. Esta é uma suposição razoável para a maioria dos programas MPI, não limitando a aplicabilidade do modelo. As contribuições de AutoElastic são:

[1] http://autoelastic.com

978-1-4673-9143-6/15/$31.00

- Infraestrutura elástica para aplicações HPC para prover **elasticidade assíncrona**, tanto não bloqueando nenhum processo, quanto instanciando ou consolidando VMs no momento de mudanças no número de processos, evitando qualquer possibilidade de finalização prematura da aplicação;
- Levando em consideração o comportamento reativo de AutoElastic, foi analisado o impacto do uso de **diferentes** *thresholds* **superior e inferior** e **quatro padrões de carga distintos** (constante, crescente, decrescente e onda) na elasticidade para executar uma aplicação com o protótipo de AutoElastic.

Este artigo descreve o modelo AutoElastic e seu protótipo, o qual foi desenvolvido para executar sobre uma nuvem privada utilizando a plataforma OpenNebula(http://opennebula.org). Em adição, são apresentados detalhes sobre a implementação de uma aplicação de integração numérica e uma discussão sobre o desempenho de AutoElastic nesse contexto. O restante deste artigo irá primeiramente introduzir os trabalhos relacionados na Seção II. A Seção III descreve AutoElastic em detalhes juntamente com seu modelo de aplicação. A metodologia de avaliação e os resultados são discutidos nas Seções IV e V. Por fim, a Seção VI enfatiza a contribuição científica do trabalho e destaca vários desafios que podem ser abordados no futuro.

II. Trabalhos Relacionados

Computação em Nuvem é focada tanto por provedores com propostas comerciais e middlewares de código aberto, quanto por iniciativas acadêmicas. Relacionado ao primeiro grupo, as iniciativas na Web oferecem a elasticidade ou manualmente quando considerando o ponto de vista do usuário [8], [9], [10] ou através da pré-configuração de mecanismos para elasticidade reativa [3], [5], [11]. Em particular, no último caso, o usuário deve configurar os *thresholds* e as ações de elasticidade, o que pode não ser trivial para um usuário sem experiência em ambientes de nuvem. Tais sistemas como Amazon EC2 (http://aws.amazon.com), Nimbus (http://www.nimbusproject.org) e Windows Azure (http://azure.microsoft.com) são exemplos dessa metodologia. Em particular, Amazon EC2 facilita a alocação e preparação de VMs mas não a configuração automática que é um requisito para uma verdadeira plataforma de nuvem elástica. Considerando os middlewares para nuvens privadas, como OpenStack (https://www.openstack.org), Eucalyptus (https://www.eucalyptus.com), OpenNebula e CloudStack (http://cloudstack.apache.org), a elasticidade é normalmente controlada manualmente, ou através de linha de comando ou através de alguma aplicação gráfica disponibilizada pela plataforma.

Iniciativas de pesquisa acadêmica procuram reduzir lacunas e/ou aprimorar as abordagens de elasticidade em nuvem. ElasticMPI propõe elasticidade em aplicações MPI através da abordagem *stop-reconfigure-and-go* [4]. Tal ação pode ter um impacto negativo, especialmente para aplicações HPC que não têm longa duração. Em adição, a abordagem de ElasticMPI faz uma alteração no código fonte da aplicação a fim de inserir políticas de monitoramento. Mao, Li e Humphrey [12] tratam com auto-escalabilidade alterando o número de VMs baseando-se em informações da carga de trabalho. Considerando que o programa possui tempo determinado para concluir cada uma de suas fases, a proposta trabalha com recursos e VMs para cumprir esses tempos corretamente. Martin et al. [13] apresentam um cenário típico de requisições a um serviço de nuvem que trabalha com um balanceador de carga. A elasticidade altera a quantidade de VMs trabalhadoras de acordo com a demanda do serviço. Na mesma abordagem do uso de um balanceador de carga e réplicas, Elastack aparece como um sistema executando sobre OpenStack para suprir sua falta de elasticidade [14].

Elasticidade é mais explorada no nível IaaS (Infrastructure as a Service) como uma abordagem reativa. Nesse sentido, os trabalhos não são uníssonos quanto ao uso de um simples *threshold* para os testes. Por exemplo, é possível notar os seguintes valores: (i) 70% [15]; (ii) 75% [16]; (iii) 80% [17]; (iv) 90% [14], [18]. Esses valores lidam com limites superiores que quando excedidos, disparam ações de elasticidade. Além disso, uma análise do estado da arte em elasticidade permite apontar algumas fraquezas nas iniciativas acadêmicas, que podem ser sumarizadas em cinco aspectos : (i) nenhuma estratégia propõe-se a avaliar se é um pico, quando atinge um *threshold* [13], [14]; (ii) necessidade de alteração no código fonte da aplicação [4], [19]; (iii) necessidade de conhecer previamente dados da aplicação antes de sua execução, como tempo de execução esperado para cada componente [4], [20]; (iv) necessidade de reconfigurar recursos com a parada da aplicação e subsequente recuperação [4]; (v) suposição de que a comunicação entre as VMs é dada a taxas constantes [21].

Observando os trabalhos mencionados, é possível destacar três deles que focam elasticidade em nuvem para aplicações HPC [4], [13], [19]. Eles têm em comum a abordagem do modelo de programação mestre-escravo. Particularmente, as iniciativas [4] e [19] são baseadas em aplicações iterativas, onde há uma redistribuição de tarefas pelo processo mestre a cada nova fase. Aplicações que não possuem um laço iterativo não podem ser adaptadas para essa abordagem, pois ele usa o identificador da iteração como um ponto de reinício da execução. Em adição, a elasticidade em [19] é gerenciada manualmente pelo usuário, obtendo dados de monitoramento utilizando o *framework* proposto pelos autores. Por fim, a proposta de solução de Martin et al. [13] é o tratamento eficaz das requisições de um servidor Web. Ele age como um delegador criando e consolidando instâncias baseado no fluxo de requisições que chegam e a carga das VMs trabalhadoras.

III. AutoElastic: Elasticidade em Nuvem para Aplicações HPC

Essa seção descreve o modelo AutoElastic, o qual analisa alternativas para as seguintes sentenças-problema:

1) Quais mecanismos são necessários para prover elasticidade transparentemente nos níveis de usuário e aplicação?
2) Considerando monitoramento de recursos e procedimentos de gerenciamento de VMs, como pode-se modelar a elasticidade para torná-la viável em aplicações do tipo HPC?

A ideia de AutoElastic é proporcionar elasticidade reativa de uma forma transparente e fácil para o usuário, que não

precisa escrever regras e ações para a elasticidade. Além disso, os usuários não precisam alterar sua aplicação paralela, não inserindo chamadas de elasticidade de uma biblioteca particular e nem modificando a aplicação para adicionar/remover recursos. Considerando a segunda questão acima mencionada, AutoElastic deve estar ciente da sobrecarga de instanciar uma VM, levando em conta esse conhecimento para oferecer esse recurso sem custos proibitivos.

A. Arquitetura

AutoElastic é um modelo de elasticidade em nuvem que opera no nível PaaS de uma plataforma de nuvem, agindo como um middleware que permite a transformação de uma aplicação paralela não elástica em uma elástica. O modelo funciona com elasticidade automática e reativa em ambas formas horizontal (gestão de réplicas de VMs) e vertical (redimensionamento da infraestrutura computacional), proporcionando alocação e consolidação de nós de computação e máquinas virtuais. Como uma proposta PaaS, AutoElastic propõe um middleware para compilar uma aplicação mestre-escravo iterativa, além de um gerenciador de elasticidade. A Figura 1 (a) mostra a iteração dos usuários com a nuvem, os quais precisam concentrar seus esforços somente no código da aplicação. O Gerenciador esconde do usuário os detalhes da escrita de regras e ações de elasticidade. A Figura 1 (b) ilustra a relação entre os processos, máquinas virtuais e nós computacionais. Considerando o escopo de nuvem, uma nuvem AutoElastic pode ser definida como segue:

- **Definição 1 - Nuvem AutoElastic**: uma nuvem modelada com m recursos computacionais homogêneos e distribuídos, em que no mínimo um deles (Nó 0) está sempre ativo. Esse nó é encarregado de executar uma VM com o processo mestre e outras c VMs com processos escravos, em que c significa o número de unidades de processamento (cores ou CPU) dentro de um nó em particular. O grão de elasticidade para cada ação se refere a um simples nó. Por fim, a qualquer momento, o número de VMs executando processos escravos é igual a $n = c \times m$.

A Figura 1 apresentada o Gerenciador AutoElastic como uma aplicação fora da nuvem, mas ele pode ser mapeado para o primeiro nó, por exemplo. Essa flexibilidade é atingida utilizando a API (*Application Programming Interface*) do pacote de software da nuvem. Levando em consideração que aplicações HPC são comumente intensivas quanto a CPU [22], optou-se pela criação de um único processo por VM e c VMs por nó de computação para explorar seu total potencial. Essa abordagem é baseada no trabalho de Lee et al. [23], em que os autores procuram explorar uma melhor eficiência em aplicações paralelas.

O usuário pode informar um SLA (*Service Level Agreement*) com a quantidade mínima e máxima de VMs permitida. Caso esse arquivo não for informado, assume-se que a quantidade máxima de VMs é o dobro da quantidade de VMs observada no lançamento da aplicação. O fato de que o Gerenciador, e não a aplicação, incrementa ou decrementa o número de recursos, provê o benefício da elasticidade assíncrona. Aqui, elasticidade assíncrona significa que a execução da aplicação e as ações de elasticidade ocorrem simultaneamente, não penalizando a aplicação com a sobrecarga da reconfiguração de recursos (alocação e desalocação). Contudo, esse assincronismo leva ao seguinte questionamento: Como podemos notificar a aplicação sobre a reconfiguração de recursos? Para isso, AutoElastic provê a comunicação entre as VMs e o Gerente utilizando uma área de memória compartilhada. Outras opções de comunicação também podem ser possíveis, incluindo NFS, middleware orientado a mensagens (tais como JMS ou AMQP) ou também espaço de tuplas (JavaSpaces, por exemplo). O uso de uma área compartilhada para interação de dados entre VMs é uma abordagem comum em nuvens privadas [8], [9], [10]. AutoElastic utiliza essa ideia para disparar ações como segue:

- Gerente escreve na área compartilhada, enquanto a aplicação lê:
  - Ação1: há um novo nó computacional com c máquinas virtuais, cada uma com um novo processo da aplicação.
  - Ação2: solicita permissão para consolidação de um nó computacional e suas máquinas virtuais.
- Um único processo da aplicação escreve na área compartilhada, enquanto o Gerente lê:
  - Ação3: dando permissão para consolidar o nó requisitado previamente.

Baseado na Ação1, os processos correntes podem iniciar trabalhando com a nova configuração de recursos (um único nó com c VMs, cada uma com um novo processo). A Ação2 é relevante pelas seguintes razões: (i) não parando a execução do processo enquanto procedimentos ou de comunicação ou de computação estão ocorrendo; (ii) garantindo que a aplicação não será abortada com a súbita interrupção de um ou mais processos. Em particular, a segunda razão é importante para aplicações MPI que executam sobre redes TCP/IP, pois elas comumente interrompem com o término prematuro de algum de seus processos. A Ação3 é normalmente tomada pelo processo mestre, que garante que a aplicação está em um estado global consistente em que processos podem ser desconectados apropriadamente. Em seguida, os processos restantes não trocam nenhuma mensagem com o nó dado. É utilizada uma área compartilhada pois isso facilita a notificação de todos os processos sobre a adição ou remoção de recursos, realizando reconfiguração do canal de comunicação de maneira simples.

AutoElastic oferece elasticidade em nuvem utilizando a técnica de replicação. Na ação de aumentar a infraestrutura, o Gerente aloca um novo nó e lança novas máquinas virtuais nele utilizando um modelo da aplicação. A inicialização de uma VM é finalizada com a execução de um processo escravo que irá realizar requisições para o processo mestre. Essa instanciação de VMs é controlada pelo Gerente e somente após sua inicialização estar completa, o Gerente notifica os outros processos através da Ação1. O procedimento de consolidação aumenta a eficiência de utilização de recursos e também provê um melhor gerenciamento do consumo de energia. Particularmente, Baliga et al. [24] afirmam que o número de VMs em um nó não é um fator influente para o consumo de energia, mas sim o fato de um nó estar ligado ou não.

Como em [11] e [16], o monitoramento de dados é realizado de forma periódica. Assim, o Gerente AutoElastic

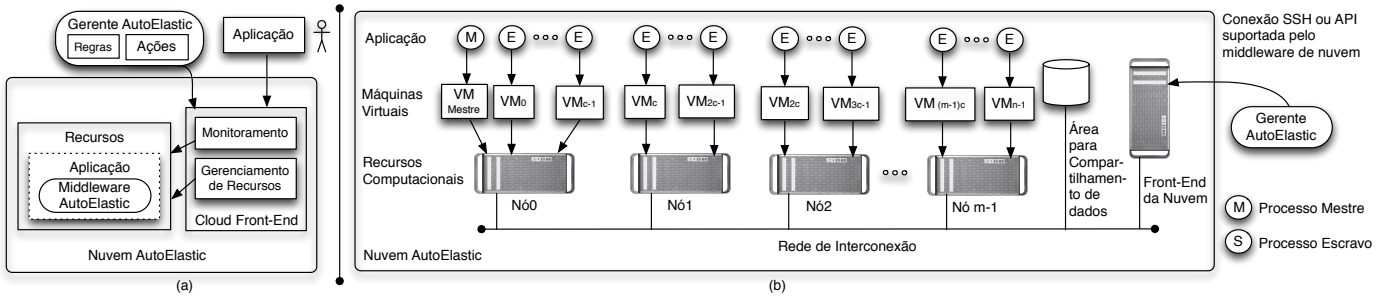

Fig. 1: Diferentes visões da arquitetura de AutoElastic: (a) passando a aplicação do usuário para a nuvem e compilando com o Middleware AutoElastic; (b) distribuição de nós, VMs e processos usando a infraestrutura de nuvem do AutoElastic, na qual cada VM engloba um único processo da aplicação e cada nó executa c VMs, em que c denota o número de CPUs do nó.

obtém a métrica CPU, aplica séries temporais baseando-se em valores passados e compara a métrica final com os thresholds superior e inferior. Mais precisamente, é empregada a técnica de Média Móvel de acordo com as Equações 1 e 2. $PC(i)$ retorna uma previsão de carga de CPU quando considerando a execução de n VMs com processos slave na observação número i do Gerente. Para realizar isso, $MM(i, j)$ informa a carga de CPU de uma máquina virtual j na observação i. A Equação 2 utiliza média móvel considerando as z observações mais recentes da carga de CPU $Carga(k, j)$ da VM j, em que $i - z \leq k \leq i$. Por fim, a Ação1 é disparada se PC for maior que o threshold superior, enquanto a Ação2 é acionada quando PC for menor que o threshold inferior.

$$PC(i) = \frac{1}{n}. \sum_{j=0}^{n-1} MM(i, j) \tag{1}$$

em que

$$MM(i, j) = \frac{\sum_{k=i-z+1}^{i} Carga(k, j)}{z} \tag{2}$$

para $i \geq z$.

B. Modelo da Aplicação Paralela

AutoElastic explora paralelismo de dados em aplicações paralelas iterativas de passagem de mensagens. Em particular, a atual versão do modelo ainda tem a restrição de operar com aplicações do estilo de programação mestre-escravo. Embora trivial, esse estilo é utilizado em várias áreas, tais como algoritmos genéticos, técnica de Monte Carlo, transformações geométricas na computação gráfica, algoritmos de criptografia e aplicações no estilo SETI-at-home [4]. Entretanto, a Ação1 permite que processos já existentes saibam os identificadores do novos e estabeleçam comunicação com eles. Outra característica é que AutoElastic trata com aplicações que não usam prazos específicos para concluir cada etapa de processamento.

Como decisão de AutoElastic, a elasticidade deve ser oferecida aos desenvolvedores sem alterar o código da aplicação. Assim, foi modelado o *framework* de comunicação analisando as interfaces tradicionais de MPI 1.x e MPI 2.x. No primeiro, a criação de processos é dada de forma estática, na qual um programa inicia e termina com o mesmo número de processos. Por outro lado, MPI 2.0 vai ao encontro das ideias de elasticidade, uma vez que habilita a criação dinâmica de novos processos e a conexão deles com os demais já existentes na topologia. As aplicações paralelas de AutoElastic são projetadas segundo o modelo MPMD (*Multiple Program Multiple Data*), no qual o mestre tem um executável e os escravos outro.

Baseado em MPI 2.0, AutoElastic trabalha com as seguintes diretivas: (i) publicar uma porta de conexão; (ii) procurar o servidor a partir de uma porta; (iii) aceitar uma conexão; (iv) requisitar uma conexão e; (v) realizar uma desconexão. Diferente da abordagem em que o processo mestre lança processos (usando a diretiva *spawn*), o modelo proposto atua segundo a outra abordagem de MPI 2.0 para o gerenciamento dinâmico de processos: comunicação ponto-a-ponto com conexão e desconexão no estilo de Sockets. O lançamento de uma VM acarreta automaticamente na execução de um processo escravo, que requisita uma conexão com o mestre. Uma aplicação com AutoElastic não necessita seguir a interface MPI 2.0, mas a semântica de cada diretiva mencionada.

A transformação de uma aplicação não elástica em uma elástica pode ser oferecida através de diferentes caminhos, todos transparentes para os usuários: (i) implementação de um programa orientado a objetos utilizando polimorfismo para sobrescrever o método para gerir a elasticidade; (ii) utilizando um tradutor de fonte-para-fonte para inserir código após a diretiva i do código do mestre; (iii) desenvolvimento de um *wrapper* em linguagens procedurais para alterar a função da diretiva i. Independente da técnica, uma região de código adicional verifica no diretório compartilhado se há novas ações.

Embora o foco inicial de AutoElastic seja aplicações mestre-escravo, a modelagem iterativa e o uso de diretivas de MPI 2.0 facilitam a inclusão de processos e o restabelecimento das conexões para uma nova topologia totalmente arbitrária. Em nível de implementação, é possível otimizar conexões e desconexões caso o processo persistir na lista de processos ativos. Essa atitude é pertinente principalmente sobre conexões TCP/IP, que usam um protocolo de três vias que sabidamente pode acarretar sobrecarga em aplicações de alto desempenho.

IV. Metodologia de Avaliação

Para a realização de testes, foi desenvolvida uma aplicação iterativa para executar na nuvem com diferentes configurações de padrões de carga e diferentes combinações de *thresholds*. Além do tempo de execução da aplicação, a ideia consiste em

analisar a reatividade da elasticidade e os custos em termos de infraestrutura para atingir um tempo de execução em particular.

A. Aplicação Paralela

A aplicação usada nos testes calcula a aproximação para a integral do polinômio $f(x)$ num intervalo fechado $[a, b]$. Para tal, foi implementado o método de Newton-Cotes para intervalos fechados conhecido como Regra do Trapézio Repetida [25]. A fórmula de Newton-Cotes pode ser útil se o valor do integrando é dada em pontos igualmente espaçados. Considere a partição do intervalo $[a, b]$ em n subintervalos iguais, cada qual de comprimento h ($[x_i, x_{i+1}]$, para $i = 0, 1, 2, ..., n-1$). Assim, $x_{i+1} - x_i = h = \frac{b-a}{n}$. Dessa forma, pode-se escrever a integral de $f(x)$ como sendo a soma das áreas dos n trapézios contidos dentro do intervalo $[a, b]$ como apresentado na Equação 3. A Equação 4 demonstra o desenvolvimento de uma integração numérica de acordo com fórmula de Newton-Cotes. Os valores x_0 e x_s na Equação 4 são iguais a a e b, respectivamente. Nesse contexto, s representa a quantidade de subintervalos. Sendo assim, existem $s + 1$ equações $f(x)$ simples para se obter o resultado final da integral numérica. O processo mestre deve distribuir essas $s + 1$ equações entre os processos escravos. Logicamente, alguns escravos podem receber mais trabalho do que outras quando $s+1$ não é inteiramente divisível pela quantidade de processos escravos. Como s define a quantidade de sub-intervalos, e consequentemente a quantidade de equações, para computar a integral, quanto maior for esse parâmetro, maior é a carga computacional para atingir o resultado final para uma equação em particular.

$$\int_a^b f(x)\,dx \approx A_0 + A_1 + A_2 + A_3 + ... + A_{s-1} \qquad (3)$$

em que A_i = area do trapezoide i, com $i = 0, 1, 2, 3, ..., s-1$.

$$\int_a^b f(x)\,dx \approx \frac{h}{2}[f(x_0) + f(x_s) + 2.\sum_{i=1}^{x-1} f(x_i)] \qquad (4)$$

B. Padrões de Carga de Processamento

A carga de trabalho recebida pelo processo mestre consiste em uma lista de equações e seus parâmetros, enquanto o retorno é a mesma quantidade de valores de integração numérica. Buscando analisar o impacto de diferentes configurações de *thresholds* na aplicação paralela com diferentes padrões de carga, o parâmetro s foi utilizado para definir quatro padrões de carga: Constante, Crescente, Decrescente e Onda. Nesse sentido, para cada conjunto de equações enviadas para processamento para o processo mestre, em cada iteração (uma equação é selecionada para cálculo) o parâmetro s foi recalculado individualmente, modelando um determinado padrão de carga. Por exemplo, s pode ser igual a $1x$ para a primeira equação, $2x$ para a segunda, $3x$ para a terceira e assim por diante para gerar o padrão Crescente. A Tabela I e a Figura 2 apresentam respectivamente as funções para o cálculo do valor de s para a geração de cada padrão de carga e o modelo de entrada de dados utilizado nos testes, em que $s = carga(x)$. O parâmetro *$iterations* na figura significa a quantidade de equações que serão geradas para o processamento, resultando no mesmo número de integrações numéricas. Ainda, o polinômio para cada uma dessas equações é o mesmo devido ao fato de não haver importância no polinômio selecionado, pois o foco real são as variações do processamento e não o resultado final da integração numérica.

TABLE I: Funções para expressar os diferentes padrões de carga. Em *carga(x)*, x é o índice da equação que será processada.

Carga	Função de Carga	Parâmetros			
		v	w	t	z
Constante	$carga(x) = \frac{w}{2}$	-	1000000	-	-
Crescente	$carga(x) = x * t * z$	-	-	0.2	500
Decrescente	$carga(x) = w - (x * t * z)$	-	1000000	0.2	500
Onda	$carga(x) = v * z * sen(t * x) + v * z + w$	1	500	0.00125	500000

```
Polynomial                +;5;x^5;+;x^2;+;x^1      +;5;x^5;+;x^2;+;x^1      +;5;x^5;+;x^2;+;x^1      +;5;x^5;+;x^2;+;x^1
$a, $b                    1,10                     1,10                     1,10                     1,10
load                      CONSTANT                 ASCENDING                DESCENDING               WAVE
$iterations               10000                    10000                    10000                    10000
$v, $w, $t, $z            0,1000000,0,0            0,0,0.2,500              0,1000000,0.2,500        1,500,0.00125,500000
(a)                       (b)                      (c)                      (d)                      (e)
```

Fig. 2: (a) Modelo de arquivo de entrada utilizado nos testes; (b), (c), (d) e (e) são instâncias do modelo observando as funções de carga da Tabela I.

A Figura 3 apresenta graficamente uma representação de cada padrão de carga. O eixo x expressa a iteração (cada iteração representa uma equação que será calculada, dividida e distribuída pelo processo mestre), enquanto o eixo y representa a respectiva carga de processamento para aquela iteração (valor de s). Novamente, a carga é definida pela quantidade de subintervalos s entre os limites a e b, que nos experimentos são 1 e 10, respectivamente. Quanto maior a quantidade de subintervalos, maior é a quantidade de equações a serem calculadas pelos processos escravos, e consequentemente maior é a carga de processamento.

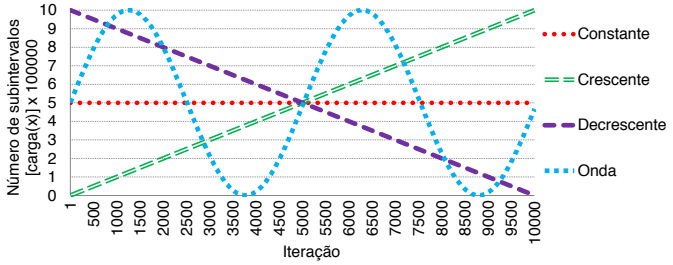

Fig. 3: Visão gráfica dos padrões de carga.

C. Cenários de Testes e Avaliação

As cargas foram executadas em dois cenários diferentes considerando a configuração inicial de recursos: (i) iniciando com 2 nós e (ii) iniciando com 4 nós. Cada carga foi executada contra cada cenário considerando AutoElastic com a elasticidade habilitada e desabilitada. No caso em que a elasticidade estava habilitada, todas as cargas foram executadas 25 vezes, em que 25 é o número total de combinações possíveis entre os *thresholds* selecionados. Assumindo as escolhas de diferentes trabalhos, nos quais podem ser encontrados *thresholds*

superiores como 50% [26], 70% [15], [26], 75% [16], 80% [18], [26] e 90% [14], [26], foram adotados 70%, 75%, 80%, 85% e 90%, enquanto que para os *thresholds* inferiores foram adotados 30%, 35%, 40%, 45% e 50%. Particularmente, o intervalo para os *thresholds* inferiores foi baseado no trabalho de Haidari et al. [26], que propõe uma análise teórica com teoria de filas para observar o desempenho de elasticidade em nuvem.

Além da perspectiva de desempenho, também foi analisado o consumo de energia a fim de perceber o impacto da elasticidade. Em outras palavras, o objetivo não é reduzir o tempo de execução da aplicação utilizando um grande número de recursos consumindo muito mais energia. Empiricamente, a Equação 5 é utilizada para estimar o consumo de energia baseando-se na quantidade de recursos utilizados. Nesse contexto, u expressa a quantidade máxima de VMs que o Gerente AutoElastic alocou durante a execução do programa. Ainda, $T(i)$ refere-se ao tempo que a aplicação foi executada utilizando i máquinas virtuais, cada qual com 1 processo escravo. Levando em consideração a medição de energia (Equação 5) e o tempo de execução da aplicação, é possível avaliar o custo $Custo$ multiplicando os dois valores mencionados (Equação 6). O objetivo final consiste em obter um custo melhor quando AutoElastic executa com a elasticidade habilitada em comparação com a execução da aplicação com um número fixo de recursos.

$$Energia = \sum_{i=1}^{u} (i \times T(i)) \tag{5}$$

$$Custo = Energia \ \times \ Tempo\_Aplic \tag{6}$$

Considerando a infraestrutura de nuvem utilizada, foi configurado um ambiente OpenNebula 4.2 em um cluster com 10 nós homogêneos. Cada nó possui um processador com dois núcleos de 2.9 GHz e 4 GB de memória RAM, interconectados em uma rede 100 Mbps.

V. Resultados

Esta seção apresenta os resultados obtidos quando executando a aplicação paralela na nuvem considerando dois cenários: (i) utilizando AutoElastic, habilitando a elasticidade; (ii) utilizando AutoElastic, apenas para monitoramento e sem a realização da elasticidade. Particularmente, os valores de *thresholds* não são importantes para o segundo cenário, visto que nenhuma operação de elasticidade é realizada.

A. Impacto dos Thresholds no Tempo de Execução da Aplicação

Com o objetivo de analisar o impacto e possíveis tendências dos *thresholds* utilizados, os resultados de todas as execuções foram organizados nas Figuras 4 e 5. Ambas apresentam o tempo final de execução da aplicação no cenário em que a elasticidade estava habilitada e as respectivas combinações de *thresholds*. Do ponto de vista de desempenho, foi observado que o tempo da aplicação não é afetado significantemente quando o *threshold* inferior varia (Figura 4 (a) e Figura 5 (a)). Por outro lado, a variação do *threshold* superior impacta diretamente no desempenho da aplicação (Figura 4 (b) e Figura 5 (b)), o qual quanto mais alto for, maior também é o tempo de execução. A falta de reatividade é a principal causa desta situação, pois a aplicação executa em estado de sobrecarga durante um longo período quando avaliando *thresholds* próximos a 90%. Particularmente, esse comportamento é mais evidente no padrão de carga Crescente. Neste caso, a carga de trabalho cresce continuamente e então, um *threshold* próximo a 70% pode alocar recursos mais rápido, aliviando a carga da CPU do sistema rapidamente.

A Tabela II apresenta os resultados dos cenários que obtiveram os menores e maiores tempos de execução da aplicação. Em adição, as Figuras 6 e 7 ilustram o desempenho envolvendo a coluna dos menores tempos obtidos nos cenários mencionados anteriormente. Cada nó inicia com duas VMs, cada uma executando um processo escravo em cada um dos dois cores do nó. Considerando que a aplicação é intensiva quanto a CPU, as execuções sem elasticidade com 4 nós iniciais superam, em média, em 65% os testes sem elasticidade com 2 nós. Isso explica também os melhores resultados obtidos com a elasticidade habilitada considerando a configuração inicial com 2 e 4 nós. Em outras palavras, a possibilidade de alterar a quantidade de recursos tem um impacto significativo quando iniciando com uma configuração mais limitada. Por exemplo, a Figura 6 (a) mostra o padrão de carga Crescente e o incremento até 12 CPUs (6 nós) com a elasticidade, denotando um ganho de desempenho de até 31%. Aqui, pode ser observado que a quantidade de CPU utilizada atinge o total de recursos alocados rapidamente, demonstrando a característica de intensividade quanto a CPU da aplicação. Considerando esta figura e o padrão de carga Decrescente, foram alocadas até 10 CPUs que vieram a ser pouco utilizadas, sendo liberadas perto do final da execução da aplicação. Isso ocorre porque AutoElastic não trabalha com conhecimento prévio da aplicação, agindo somente com dados capturados em tempo de execução.

B. Consumo de Energia e Custo

As Figuras 8 e 9 apresentam perfis da execução da aplicação, que descrevem o mapeamento de VMs quando são considerados os menores e maiores tempos apresentados na Tabela II. O cenário em que se inicia a execução com 2 nós (4 VMs) não apresenta ações de elasticidade no pior caso quando utilizando o *threshold* superior igual a 90%. Isso explica os resultados na parte (b) da Figura 8. Ainda, com 2 nós iniciais, um *threshold* superior igual a 70% é responsável por alocar até 12 VMs na carga Crescente como pode ser observado na Figura 8 (a). Contrariamente a essa situação, a Figura 9 mostra menos variação na configuração de recursos, quando 4 nós (8 VMs) são mantidos na maior parte do tempo da execução. Como exceção, pode ser observado a carga Crescente como pior caso, em que um *threshold* superior igual a 90% é empregado. A aplicação inicia a redução da quantidade de VMs de 8, para 6 até 4, utilizando essa última configuração em 96% do tempo de execução. Embora não seja superior a uma carga de 90%, não ampliando a infraestrutura, a alocação de mais recursos nesta situação poderia ajudar tanto no balanceamento de carga entre as CPUs como reduzindo o tempo de aplicação.

A Figura 10 apresenta o total de CPU alocada e o total de CPU utilizada considerando os menores tempos (ver Tabela II) de ambos cenários com 2 e 4 nós iniciais. Como pode ser

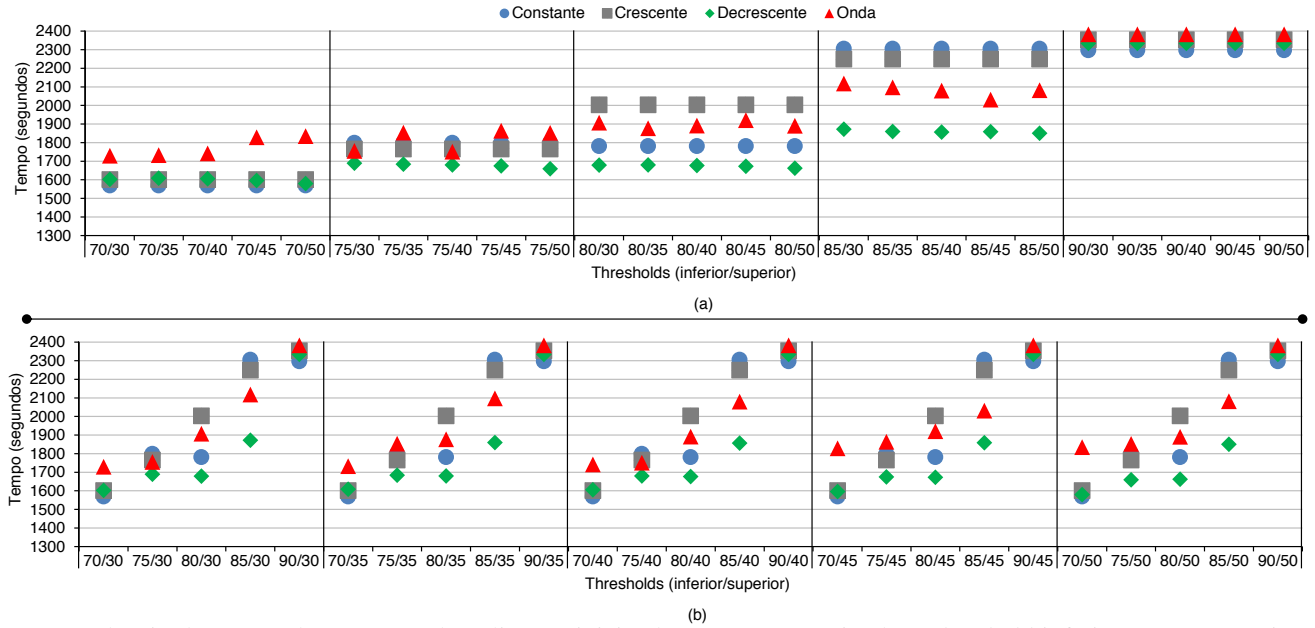

Fig. 4: Tendência do tempo de execução da aplicação iniciando com 2 nós variando o *threshold* inferior (a) e o superior (b).

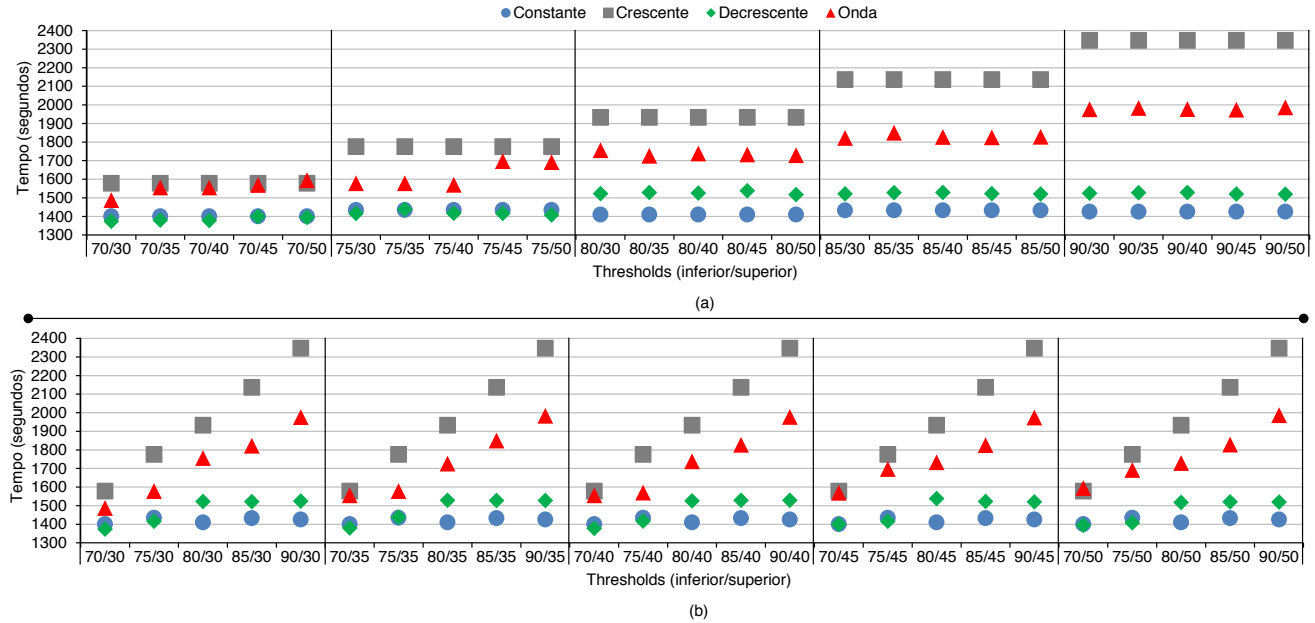

Fig. 5: Tendência do tempo de execução da aplicação iniciando com 4 nós variando o *threshold* inferior (a) e o superior (b).

TABLE II: Combinações de *thresholds* que obtiveram o menor e maior tempo de execução da aplicação para cada padrão de carga considerando o cenário inicial de recursos e a disponibilidade de elasticidade.

Modo	Nós iniciais/ CPUs	Padrão de Carga	Menores tempos					Maiores tempos				
			Thresholds		Tempo (segundos)	Energia (Equação 5)	Custo (Equação 6)	Thresholds		Tempo (segundos)	Energia (Equação 5)	Custo (Equação 6)
			Superior	Inferior				Superior	Inferior			
AutoElastic com elasticidade	2/4	Crescente	70	Todos	1601	11160	17867160	90	Todos	2354	9416	22165264
		Constante	70	Todos	1569	11206	17582214	85	Todos	2305	9220	21252100
		Decrescente	70	50	1580	12014	18982120	90	Todos	2334	9336	21790224
		Onda	70	30	1730	12518	21656140	90	Todos	2383	9532	22714756
	4/8	Crescente	70	Todos	1578	11268	17780904	90	Todos	2348	9632	22615936
		Constante	70	Todos	1401	11208	15702408	75	Todos	1435	11480	16473800
		Decrescente	70	30	1374	14184	19488816	80	45	1538	11636	17896168
		Onda	70	30	1487	14082	20939934	90	50	1987	10578	21018486
AutoElastic sem elasticidade	2/4	Crescente	-	-	2317	9268	21473956	-	-	-	-	-
		Constante	-	-	2281	9124	20811844	-	-	-	-	-
		Decrescente	-	-	2308	9232	21307456	-	-	-	-	-
		Onda	-	-	2345	9380	21996100	-	-	-	-	-
	4/8	Crescente	-	-	1510	12080	18240800	-	-	-	-	-
		Constante	-	-	1384	11072	15323648	-	-	-	-	-
		Decrescente	-	-	1506	12048	18144288	-	-	-	-	-
		Onda	-	-	1544	12352	19071488	-	-	-	-	-

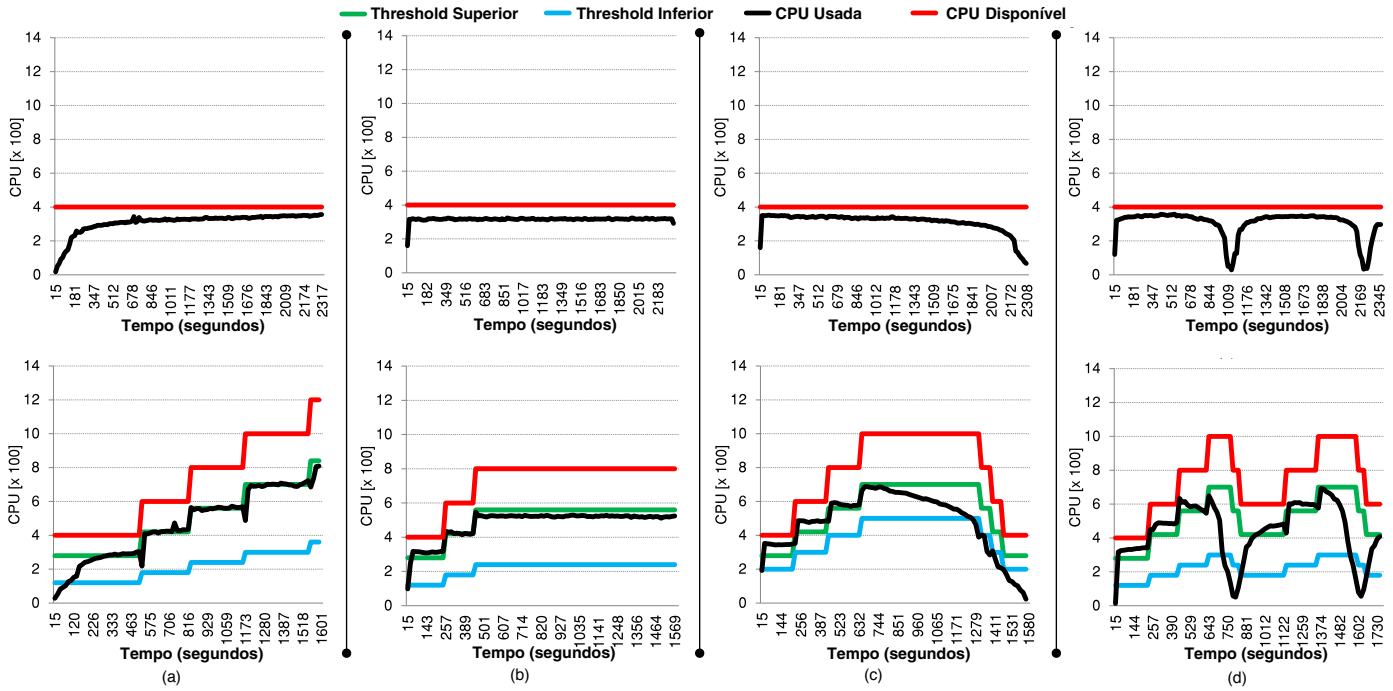

Fig. 6: Histórico de utilização de recursos das execuções que obtiveram os menores tempos iniciadas com 2 nós apresentadas na Tabela II. A parte superior se refere à execução sem elasticidade, enquanto que a parte inferior se refere à execução com a elasticidade habilitada. Cada coluna representa um padrão de carga: (a) Crescente; (b) Constante; (c) Decrescente; (d) Onda.

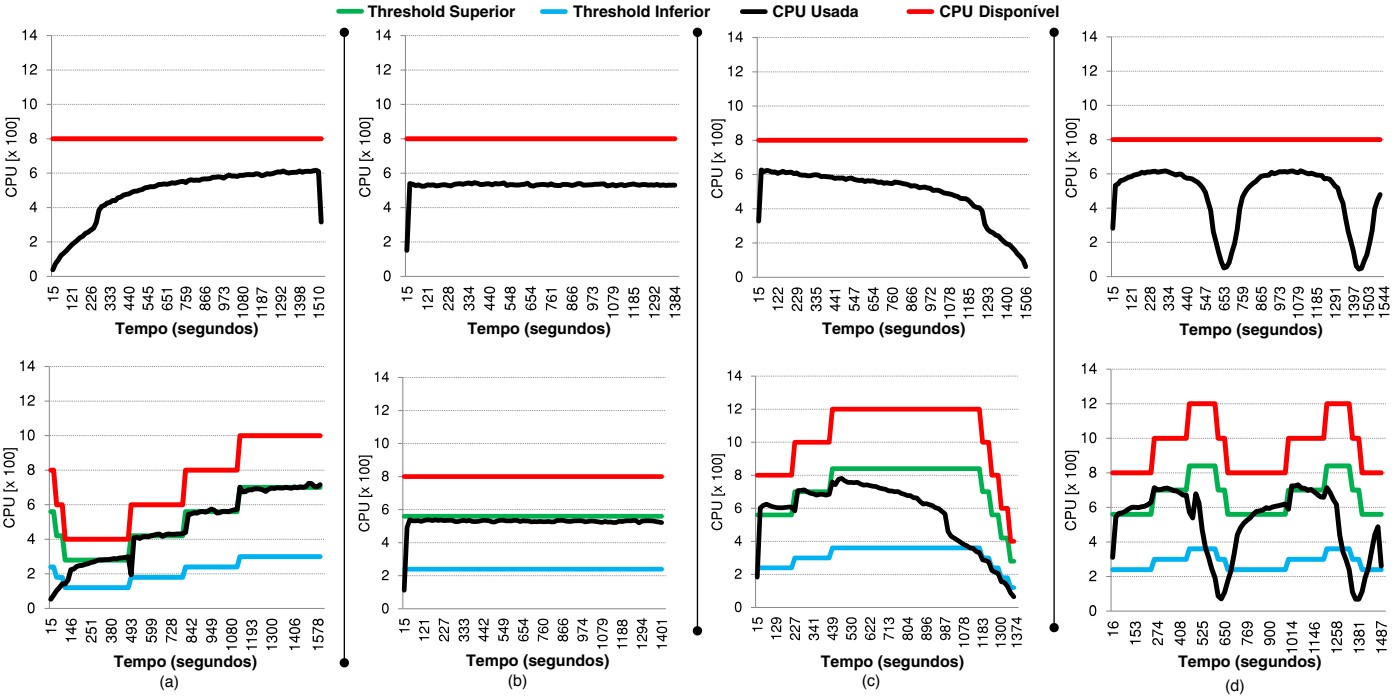

Fig. 7: Histórico de utilização de recursos das execuções que obtiveram os menores tempos iniciadas com 4 nós apresentados na Tabela II. A parte superior se refere à execução sem elasticidade, enquanto que a parte inferior se refere à execução com a elasticidade habilitada. Cada coluna representa um padrão de carga: (a) Crescente; (b) Constante; (c) Decrescente; (d) Onda.

visto, todas as cargas utilizaram menos CPU com a elasticidade habilitada, exceto a carga Constante iniciando com 4 nós que obteve resultado igual com a elasticidade habilitada e desabilitada. Contudo, o uso de 2 nós iniciais implica em alocar uma maior quantidade de CPU quando a elasticidade está habilitada. Esse comportamento é esperado por duas razões: (i) AutoElastic não usa informações prévias referentes à aplicação; (ii) após alocar recursos, toda a carga geral do

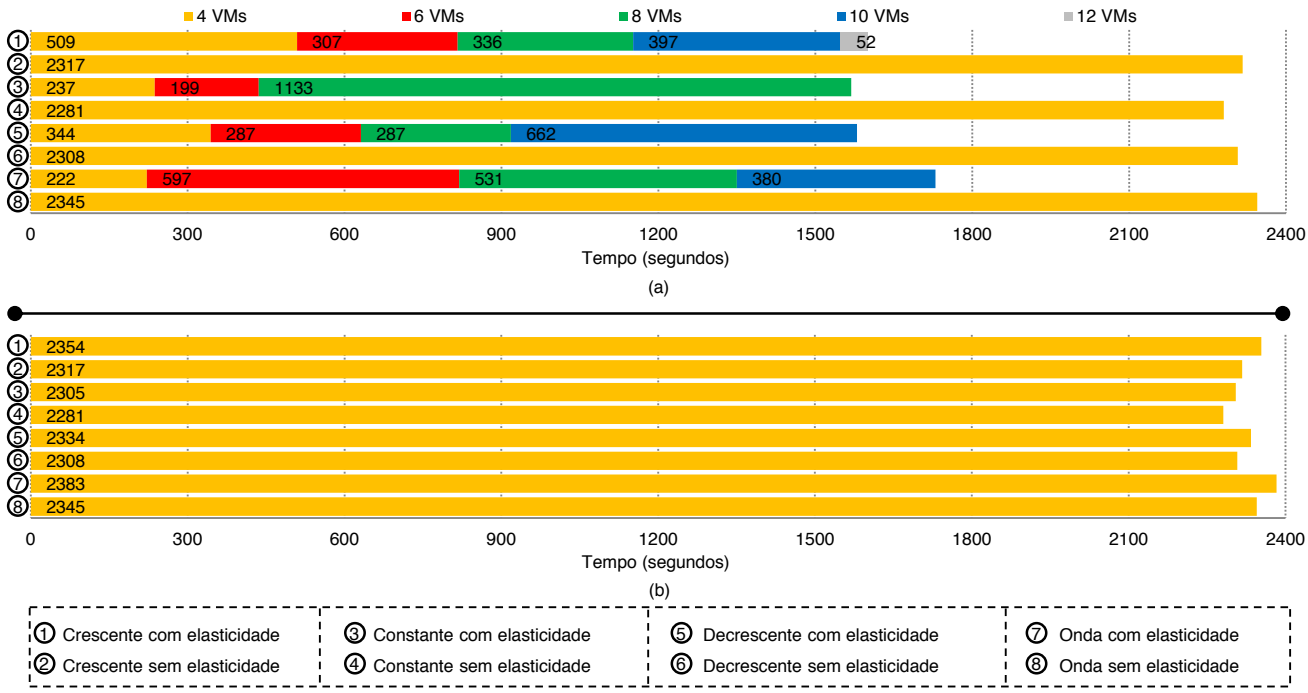

Fig. 8: Profile das execuções que obtiveram os menores (a) e maiores (b) tempos de execução da aplicação iniciando com 2 nós para cada padrão de carga com e sem elasticidade.

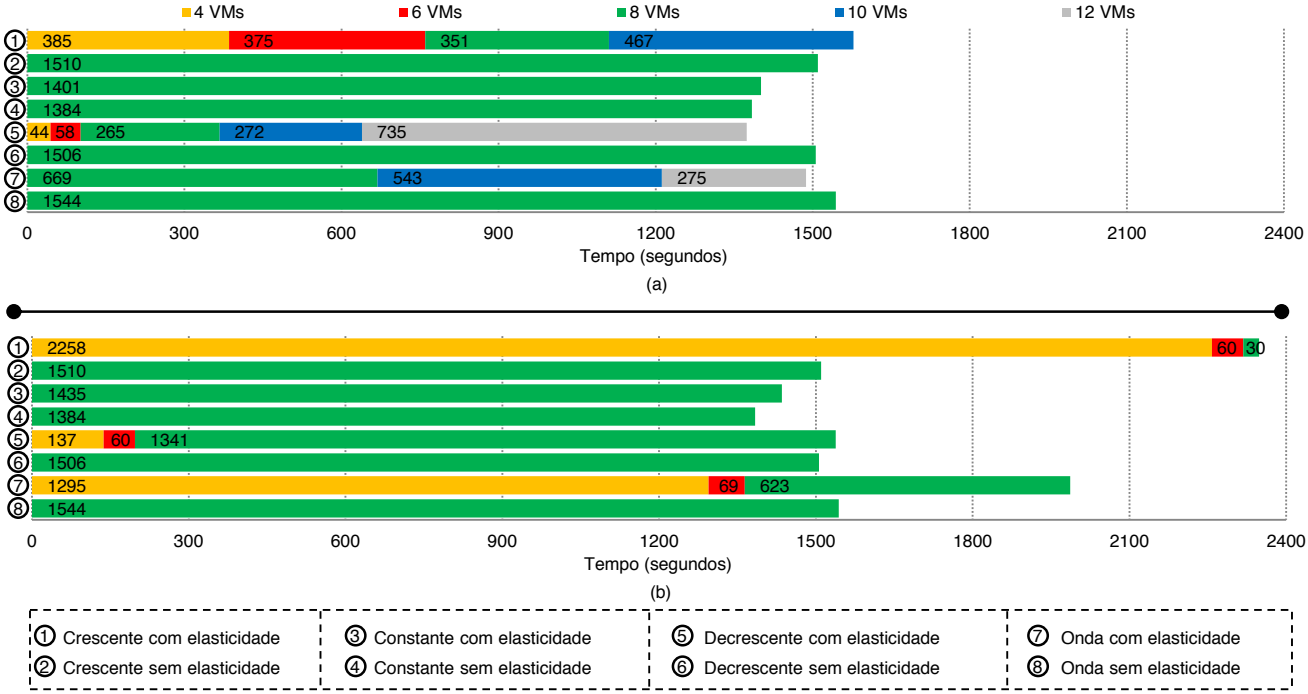

Fig. 9: Profile das execuções que obtiveram os menores (a) e maiores (b) tempos de execução da aplicação iniciando com 4 nós para cada padrão de carga com e sem elasticidade.

ambiente diminui resultando em um melhor balanceamento de carga mas em uma pior utilização dos recursos.

Considerando as Figuras 8 (a) e 9 (a), foi computado o consumo de energia como declarado na Equação 5. Considerando esta métrica e o tempo da aplicação, é possível calcular o custo conforme a Equação 6. A ideia final referente ao custo é sumarizada na Equação 7, em que $s1$ significa o cenário com AutoElastic e elasticidade habilitada e $s2$, o cenário com AutoElastic e elasticidade desabilitada. Considerando esta equação, o objetivo é ou reduzir o tempo evitando grande penalidade de recursos para isso ou apresentar um tempo

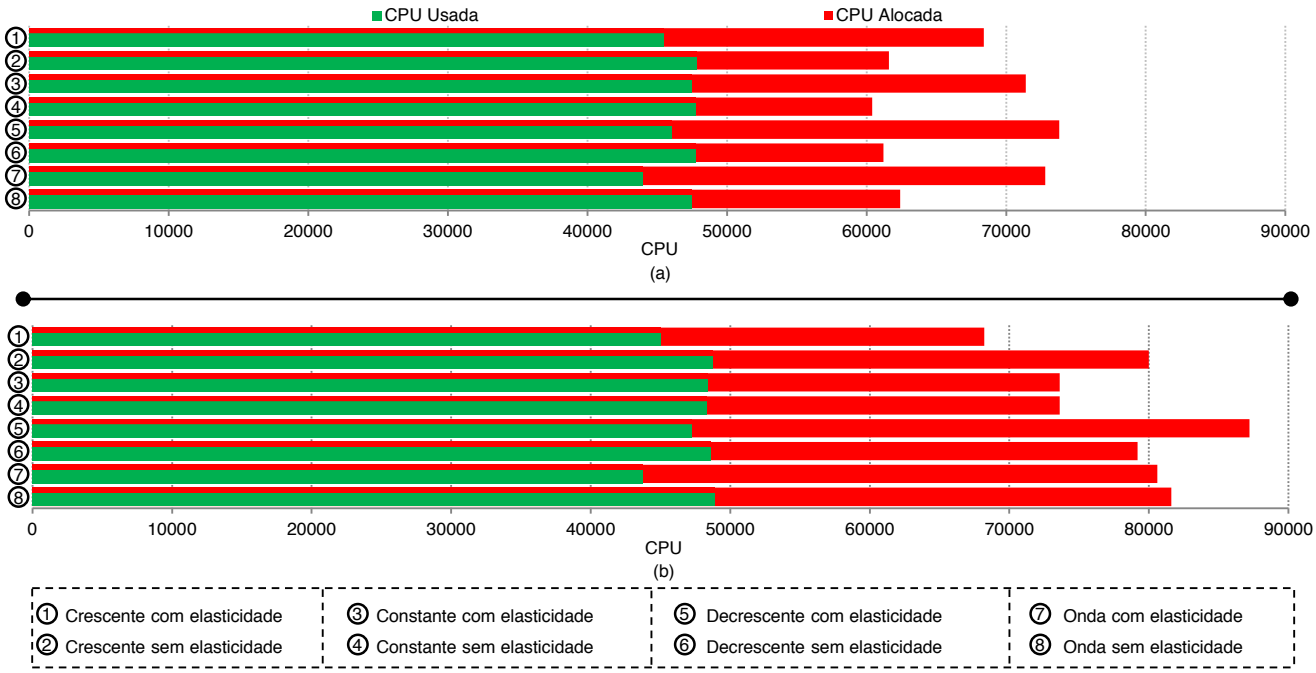

Fig. 10: Eficiência obtida pelas execuções que obtiveram os menores tempos de execução com e sem elasticidade: (a) iniciando com 2 nós e (b) iniciando com 4 nós.

ligeiramente maior que a execução não elástica mas melhorando a utilização de recursos. A Figura 11 ilustra os custos obtidos nos cenários com 2 e 4 nós iniciais. A elasticidade é responsável pelos melhores resultados no primeiro caso para todas as cargas avaliadas. Por outro lado, a falta de reatividade e *thresholds* fixos foram a principal razão para os resultados no segundo caso. Em outras palavras, a aplicação executa ineficientemente durante um longo período de tempo até atingir o *threshold*. A previsão do comportamento da aplicação e *thresholds* adaptáveis poderiam ajudar a melhorar o desempenho da aplicação quando utilizando configurações com um grão de computação reduzido (índice que pode ser estimado dividindo o trabalho envolvido nas tarefas de computação pelos custos de comunicação de rede).

$$Energia_{s1} \times TempoAplic_{s1} < Energia_{s2} \times TempoAplic_{s2} \quad (7)$$

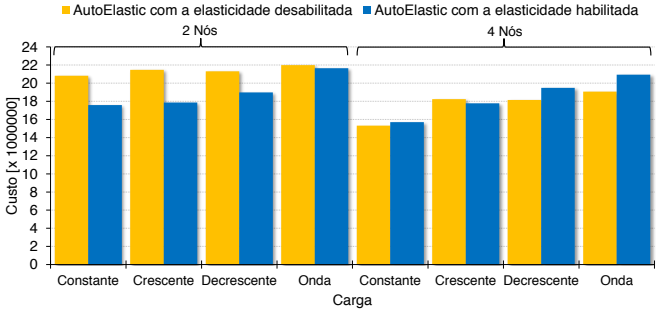

Fig. 11: Custo (Equação 6) das execuções que obtiveram os menores tempos de execução com e sem elasticidade.

VI. Conclusão

Este artigo apresentou o modelo AutoElastic, enfatizando seu funcionamento no escopo de aplicações HPC variando tanto os *thresholds* quanto o padrão de carga da aplicação. Considerando as questões listadas na Seção III, AutoElastic age em nível de middleware objetivando aplicações de passagem de mensagens com paralelismo explícito, as quais utilizam diretivas para enviar e receber dados, bem como para conectar processos que se comunicam. Particularmente, foi adotada esta decisão de projeto devido ao fato da fácil implementação destas diretivas em MPI 2.0, o qual oferece uma API de programação no estilo *sockets* para criação dinâmica de processos. Além disso, considerando os requisitos de tempo de aplicações HPC, foi modelado um *framework* para habilitar uma nova funcionalidade denotada elasticidade assíncrona, em que a alocação e a consolidação de VMs ocorre em paralelo com a execução da aplicação.

A principal contribuição deste trabalho é a análise conjunta de um modelo de elasticidade e uma aplicação HPC ao variar ambos os parâmetros de elasticidade e padrões de carga. Assim, a discussão aqui pode ajudar programadores a configurar *thresholds* de elasticidade para explorar melhor desempenho e utilização de recursos em demandas dirigidas por CPU. Os resultados mostraram que a eficácia da elasticidade depende: (i) do grão computacional da aplicação; (ii) da reatividade da elasticidade. Assim, foi demonstrado que uma aplicação intensiva quanto ao uso de CPU pode executar mais rápido com a elasticidade em comparação com outros cenários em que a quantidade de recursos é fixa e/ou reduzida. Além disso, foi observado que um *threshold* superior (que conduz o alargamento da infraestrutura) próximo a 100% pode ser visto como uma má escolha devido a aplicação executar desnecessariamente com recursos sobrecarregados até atingir

este limite. Nos testes realizados, o menor valor para este parâmetro foi 70%, o qual resultou nos melhores desempenhos considerando todos os padrões de carga.

Trabalhos futuros consistem em estudar *thresholds* adaptáveis a fim de gerar melhor reatividade nas ações elásticas, considerando ambas métricas de desempenho e energia. Embora a aplicação de integral numérica ser útil para avaliar as contribuições de AutoElastic, pretende-se explorar a elasticidade em aplicações irregulares e em *benchmarks* tradicionais de HPC [27]. Por fim, planeja-se também estender AutoElastic para para cobrir elasticidade em outros modelos de programação paralela, tais como divisão-e-conquista, *pipeline* e *bulk-synchronous parallel* (BSP).

AGRADECIMENTOS

Este trabalho foi parcialmente suportado pelos seguintes órgãos brasileiros de fomento: CAPES, CNPq e FAPERGS. Em adição, os autores também agradecem ao grupo Santander pelo apoio e patrocínio.

REFERENCES

[1] M. Mohan Murthy, H. Sanjay, and J. Anand, "Threshold based auto scaling of virtual machines in cloud environment," in *Network and Parallel Computing*, ser. Lecture Notes in Computer Science, C.-H. Hsu, X. Shi, and V. Salapura, Eds. Springer Berlin Heidelberg, 2014, vol. 8707, pp. 247–256.

[2] J. Bao, Z. Lu, J. Wu, S. Zhang, and Y. Zhong, "Implementing a novel load-aware auto scale scheme for private cloud resource management platform," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*, May 2014, pp. 1–4.

[3] S. Sah and S. Joshi, "Scalability of efficient and dynamic workload distribution in autonomic cloud computing," in *Issues and Challenges in Intelligent Computing Techniques (ICICT), 2014 International Conference on*, Feb 2014, pp. 12–18.

[4] A. Raveendran, T. Bicer, and G. Agrawal, "A framework for elastic execution of existing mpi programs," in *Proceedings of the 2011 IEEE Int. Symposium on Parallel and Distributed Processing Workshops and PhD Forum*, ser. IPDPSW '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 940–947.

[5] P. Jamshidi, A. Ahmad, and C. Pahl, "Autonomic resource provisioning for cloud-based software," in *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS 2014. New York, NY, USA: ACM, 2014, pp. 95–104. [Online]. Available: http://doi.acm.org/10.1145/2593929.2593940

[6] E. F. Coutinho, G. Paillard, and J. N. de Souza, "Performance analysis on scientific computing and cloud computing environments," in *Proceedings of the 7th Euro American Conference on Telematics and Information Systems*, ser. EATIS '14. New York, NY, USA: ACM, 2014, pp. 5:1–5:6. [Online]. Available: http://doi.acm.org/10.1145/2590651.2590656

[7] R. R. Expósito, G. L. Taboada, S. Ramos, J. Touriño, and R. Doallo, "Evaluation of messaging middleware for high-performance cloud computing," *Personal Ubiquitous Comput.*, vol. 17, no. 8, pp. 1709–1719, Dec. 2013. [Online]. Available: http://dx.doi.org/10.1007/s00779-012-0605-3

[8] B. Cai, F. Xu, F. Ye, and W. Zhou, "Research and application of migrating legacy systems to the private cloud platform with cloudstack," in *Automation and Logistics (ICAL), 2012 IEEE International Conference on*, aug. 2012, pp. 400 –404.

[9] D. Milojicic, I. M. Llorente, and R. S. Montero, "Opennebula: A cloud management tool," *Internet Computing, IEEE*, vol. 15, no. 2, pp. 11 –14, march-april 2011.

[10] X. Wen, G. Gu, Q. Li, Y. Gao, and X. Zhang, "Comparison of open-source cloud management platforms: Openstack and opennebula," in *Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on*, may 2012, pp. 2457 –2461.

[11] D. Chiu and G. Agrawal, "Evaluating caching and storage options on the amazon web services cloud," in *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*, oct. 2010, pp. 17 –24.

[12] M. Mao, J. Li, and M. Humphrey, "Cloud auto-scaling with deadline and budget constraints," in *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*, oct. 2010, pp. 41 –48.

[13] P. Martin, A. Brown, W. Powley, and J. L. Vazquez-Poletti, "Autonomic management of elastic services in the cloud," in *Proceedings of the 2011 IEEE Symposium on Computers and Communications*, ser. ISCC '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 135–140.

[14] L. Beernaert, M. Matos, R. Vilaça, and R. Oliveira, "Automatic elasticity in openstack," in *Proceedings of the Workshop on Secure and Dependable Middleware for Cloud Monitoring and Management*, ser. SDMCMM '12. New York, NY, USA: ACM, 2012, pp. 2:1–2:6. [Online]. Available: http://doi.acm.org/10.1145/2405186.2405188

[15] W. Dawoud, I. Takouna, and C. Meinel, "Elastic vm for cloud resources provisioning optimization," in *Advances in Computing and Communications*, ser. Communications in Computer and Information Science, A. Abraham, J. Lloret Mauri, J. Buford, J. Suzuki, and S. Thampi, Eds. Springer Berlin Heidelberg, 2011, vol. 190, pp. 431–445.

[16] S. Imai, T. Chestna, and C. A. Varela, "Elastic scalable cloud computing using application-level migration," in *Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing*, ser. UCC '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 91–98.

[17] M. Mihailescu and Y. M. Teo, "The impact of user rationality in federated clouds," *Cluster Computing and the Grid, IEEE International Symposium on*, vol. 0, pp. 620–627, 2012.

[18] B. Suleiman, "Elasticity economics of cloud-based applications," in *Proceedings of the 2012 IEEE Ninth International Conference on Services Computing*, ser. SCC '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 694–695.

[19] D. Rajan, A. Canino, J. A. Izaguirre, and D. Thain, "Converting a high performance application to an elastic cloud application," in *Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science*, ser. CLOUDCOM '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 383–390.

[20] T. Knauth and C. Fetzer, "Scaling non-elastic applications using virtual machines," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, july 2011, pp. 468 –475.

[21] X. Zhang, Z.-Y. Shae, S. Zheng, and H. Jamjoom, "Virtual machine migration in an over-committed cloud," in *Network Operations and Management Symposium (NOMS), 2012 IEEE*, april 2012, pp. 196 – 203.

[22] F. Azmandian, M. Moffie, J. Dy, J. Aslam, and D. Kaeli, "Workload characterization at the virtualization layer," in *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on*, July 2011, pp. 63–72.

[23] Y. Lee, R. Avizienis, A. Bishara, R. Xia, D. Lockhart, C. Batten, and K. Asanovic, "Exploring the tradeoffs between programmability and efficiency in data-parallel accelerators," in *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*, 2011, pp. 129–140.

[24] J. Baliga, R. Ayre, K. Hinton, and R. Tucker, "Green cloud computing: Balancing energy in processing, storage, and transport," *Proceedings of the IEEE*, vol. 99, no. 1, pp. 149–167, 2011.

[25] M. Comanescu, "Implementation of time-varying observers used in direct field orientation of motor drives by trapezoidal integration," in *Power Electronics, Machines and Drives (PEMD 2012), 6th IET International Conference on*, 2012, pp. 1–6.

[26] F. Al-Haidari, M. Sqalli, and K. Salah, "Impact of cpu utilization thresholds and scaling size on autoscaling cloud resources," in *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, vol. 2, Dec 2013, pp. 256–261.

[27] L. Jin, D. Cong, L. Guangyi, and Y. Jilai, "Short-term net feeder load forecasting of microgrid considering weather conditions," in *Energy Conference (ENERGYCON), 2014 IEEE International*, May 2014, pp. 1205–1209.