Explorando a Elasticidade Assíncrona em Nuvem para Aplicações Paralelas Iterativas

Vinicius Rodrigues[1], **Cristiano Costa**[1], **Rodrigo da Rosa Righi**[1], **Diego Kreutz**[2]

[1]Prog. Interdisciplinar de Pós-Graduação em Computação Aplicada, Unisinos - Brasil
Email: viniciusfacco@live.com, {cac,rrrighi}@unisinos.br

[2]LaSIGE, FCUL - Portugal
Email: kreutz@lasige.di.fc.ul.pt

Abstract. *Elasticity is undoubtedly one of the most known capabilities related to cloud computing. In the high performance computing area, initiatives normally use bag-of-tasks applications requiring changes in the source code in order to address elasticity. In this context, this article presents a elasticity model called AutoElastic. AutoElastic acts at middleware level over iterative parallel applications, offering automatic resources provisioning. Its differential approach appears on the asynchronous elasticity concept. Besides the model itself, the article also presents a prototype built with OpenNebula and its evaluation with an iterative parallel application, showing performance gains of up to 14% and a low intrusivity.*

Resumo. *A elasticidade é sem dúvida uma das características mais marcantes da computação em nuvem. Na área de computação de alto desempenho, as iniciativas normalmente trabalham aplicações no estilo sacola-de-tarefas com necessidade de alterações no código para o tratamento da elasticidade. Nesse contexto, esse artigo apresenta o modelo de elasticidade chamado AutoElastic. AutoElastic atua em nível de middleware sobre aplicações paralelas iterativas, oferecendo provisionamento automático de recursos. Seu diferencial está no conceito de elasticidade assíncrona. Além do modelo, o presente artigo também apresenta um protótipo construído com OpenNebula e sua avaliação com uma aplicação iterativa, demonstrando ganhos em relação a tempo de até 14% e baixa intrusividade.*

1. Introdução

Uma das características mais importantes que distinguem a computação em nuvem de outras abordagens de sistemas distribuídos é a elasticidade [Dawoud et al. 2011, Kouki et al. 2014]. Em particular, a elasticidade de recursos em ambientes de nuvem (*cloud computing*) explora o fato que a alocação de recursos é um procedimento que pode ser efetuado dinamica e automaticamente de acordo com a demanda do serviço ou do usuário. Apesar dos benefícios como a melhora no desempenho e a redução de custos e riscos, a elasticidade também impõe desafios para o desenvolvimentro de aplicações e serviços. Esforços recentes mostram a exploração da elasticidade em nuvem para serviços com alta demanda de entrada/saída, como tratamento de vídeo pela Internet, lojas online de venda de produtos, aplicações BOINC, governança eletrônica e Web Services [Sinha and Khreisat 2014]. A abordagem mais comum em tais serviços é

a elasticidade reativa baseada na replicação de máquinas virtuais quando um determinado índice de observação (*threshold*) de uma métrica ou combinação delas for atingido [Raveendran et al. 2011]. Assim, o balanceador de carga mantido pelo próprio provedor de nuvem gerencia as demandas e despacha cada uma para a réplica mais apta [Dawoud et al. 2011, Imai et al. 2012, Mao et al. 2010].

Múltiplos provedores de computação em nuvem estão focando em aplicações que demandam alto desempenho (HPC). Entretanto, aplicações de HPC na sua maioria possuem dificuldade para usufruir da elasticidade visto que normalmente são projetadas com um número fixo de processos [Sinha and Khreisat 2014]. Essa é a situação daqueles programas que seguem a interface 1.0 de MPI (*Message Passing Interface*). MPI 2.0 sobrepassa essa limitação proporcionando a criação de processos em tempo de execução. Para exploração da elasticidade em nível de aplicação com MPI 2.0, é necessário um esforço manual para mudar o grupo de processos e redistribuir os dados eficientemente para usar um número diferente de processos [Raveendran et al. 2011, Goh and Tan 2014]. Para efetivar essa ideia, um ou mais processos devem proativamente ou de forma periódica analisar se há novos recursos para assim utilizá-lo e proceder com o mecanismo de balanceamento de carga. Ainda, a remoção de uma máquina virtual com baixo índice de uso de CPU pode levar ao término prematuro da aplicação, uma vez que essa é uma prática comum na bibliotecas de programação para aplicações fortemente acopladas.

Nesse contexto, esse artigo apresenta o modelo **AutoElastic**[1], que gerencia a elasticidade em aplicações HPC iterativas. AutoElastic atua em nível de middleware, ou seja, na camada PaaS de uma nuvem, não impondo modificações no código fonte da aplicação tampouco precisando de informações prévias sobre o seu comportamento. O modelo trabalha com aplicações do tipo iterativas, caracterizadas com *timesteps* ou *loops*, dado que representa um estilo largamente difundido para a construção de aplicações paralelas. Em termos de contribuição científica, AutoElastic oferece elasticidade assíncrona através de um arcabouço em que os processos não ficam bloqueados nas operações de alocação e desalocação de máquinas virtuais, ou VMs. Isso tem um impacto significativo no dueto HPC e elasticidade, uma vez que aplicações paralelas são sensíveis a interferências que possam piorar o desempenho. Esse artigo descreve AutoElastic e um protótipo construído com o middleware OpenNebula. Testes com uma aplicação científica mostram ganhos de desempenho de até 14% quando usado AutoElastic, na comparação com provisionamento fixo. Na sequência do artigo, são descritos os trabalhos relacionados. As Seções 3 e 4 apresentam AutoElastic e seu protótipo. A metodologia de avaliação e os resultados estão nas Seções 5 e 6. Por fim, a Seção 7 apresenta a conclusão e a contribuição científica.

2. Trabalhos Relacionados

O tema computação em nuvem é abordado tanto por provedores com intuito comercial e middlewares com código aberto, quanto por trabalhos acadêmicos. Quanto ao primeiro grupo, os sistemas disponíveis na Web se destacam por oferecer o tratamento da elasticidade de forma manual para o usuário [Cai et al. 2012, Milojicic et al. 2011, Wen et al. 2012] ou através de preconfiguração de mecanismos com elasticidade reativa [Chiu and Agrawal 2010, Roloff et al. 2012]. Em particular, nesse último caso, o usuário deve definir *thresholds* e ações de elasticidade, o que pode não ser tri-

[1]Página Web do projeto: http://autoelastic.github.io/autoelastic

vial para usuários não especialistas nesse tipo de ambiente. Sistemas como Amazon AWS (http://aws.amazon.com), Nimbus (http://www.nimbusproject.org) e Windows Azure (http://azure.microsoft.com) são exemplos dessa metodologia. Quanto a middlewares para nuvens privadas como o OpenStack (https://www.openstack.org), OpenNebula (http://opennebula.org), Eucalyptus (https://www.eucalyptus.com) e CloudStack (http://cloudstack.apache.org), a elasticidade é normalmente controlada de forma manual, seja ela em linha de comando ou com um aplicativo gráfico de controle da infraestrutura.

Iniciativas de pesquisa acadêmica buscam sanar lacunas e/ou aprimorar abordagens para o tratamento da elasticidade. ElasticMPI propõe a elasticidade em aplicações MPI através da parada e relançamento delas no momento da reconfiguração de recursos [Raveendran et al. 2011]. Tal ação pode ter um impacto negativo, em especial para aquelas aplicações HPC que não possuem longa duração. Em adição, a abordagem de ElasticMPI faz uma alteração no código fonte da aplicação de modo a inserir diretivas de monitoramento. Ming, li e Humphrey [Mao et al. 2010] tratam a auto-escalabilidade com a alteração do número de instâncias de VMs baseado em informações da carga de trabalho. Uma vez que o programa possui *deadlines* para execução de suas fases, a proposta trabalha com recursos de VMs e nós para cumprir o prazo. Martin et al. [Martin et al. 2011] apresentam um cenário típico de requisições sobre um serviço em nuvem que atua com um balanceador de carga. Nessa mesma linha, Elastack aparece como um sistema que executa sobre OpenStack para suprir a carência de elasticidade desse último [Beernaert et al. 2012].

A elasticidade é mais explorada em nível de IaaS e de forma reativa. Nesse sentido, os trabalhos não são uníssonos quanto ao emprego de um *threshold* de carga único para os testes. Por exemplo, é possível notar os seguintes valores: (i) 70% [Dawoud et al. 2011]; (ii) 75% [Imai et al. 2012]; (iii) 80% [Mihailescu and Teo 2012]; (iv) 90% [Beernaert et al. 2012]. Em adição, a análise do estado-da-arte em elasticidade permite apontar alguns pontos fracos de iniciativas da academia. São eles: (i) não há tratamento para analisar se é um pico ao atingir um *threshold* [Beernaert et al. 2012, Martin et al. 2011]; (ii) necessidade de alteração do código fonte da aplicação [Rajan et al. 2011, Raveendran et al. 2011]; (iii) necessidade de saber dados da aplicação antes de sua execução, tais como o tempo esperado de execução de cada componente [Michon et al. 2012, Raveendran et al. 2011]; (iv) reconfiguração de recursos com parada da aplicação e posterior relançamento [Raveendran et al. 2011]; (v) suposição que a comunicação entre VMs é dada a uma taxa constante [Zhang et al. 2012].

De forma pontual, três trabalhos abordam a elasticidade para aplicações que visam desempenho em nuvem [Martin et al. 2011, Rajan et al. 2011, Raveendran et al. 2011]. Eles têm em comum o fato de abordarem modelo de programação mestre-escravo. Ambas iniciativas [Rajan et al. 2011, Raveendran et al. 2011] se baseiam em aplicações iterativas, nas quais em cada nova fase há uma redistribuição de tarefas pela entidade mestre. Em particular, a elasticidade em [Rajan et al. 2011] é gerida de forma manual pelo usuário, que obtém dados de monitoramento através do arcabouço proposto pelos autores. Apesar das lacunas já mostradas de ElasticMPI [Raveendran et al. 2011], o ponto forte desse sistema é oferecer elasticidade, ainda que com limitações, para sistemas MPI (usando um pré-compilador fonte-para-fonte). Por fim, a finalidade da solução de Martin et al. [Martin et al. 2011] é o tratamento eficiente de requisições por um servidor Web

Services. Ele atua como um delegator criando e consolidando instâncias com base no fluxo de requisições de chegada e na carga das VMs trabalhadoras.

3. AutoElastic: Modelo para Tratamento de Elasticidade em Aplicações Paralelas Iterativas

Essa seção descreve o modelo AutoElastic, que analisa alternativas para duas sentenças-problema: (i) *Quais mecanismos são necessários para oferecer elasticidade para aplicações de alto desempenho, oferecendo transparência desse recurso tanto para o usuário quanto na escrita da própria aplicação?*; (ii) *Quais aplicações de alto desempenho podem fazer uso da elasticidade em nuvem e como e sob quais restrições ela pode ser suportada?* AutoElastic oferece a capacidade de elasticidade reativa para aplicações paralelas sem a intervenção do programador ou usuário, não impondo que estes escrevam ações e regras para a elasticidade. A Figura 1 ilustra essa ideia. Além da transparência para o usuário, há também aquela em nível de aplicação, uma vez que não são necessárias linhas adicionais para coleta de dados de monitoramento ou para aumentar ou diminuir recursos. AutoElastic deve ser ciente do tempo de lançamento de uma VM e deve trabalhar de modo que essa sobrecarga impacte o mínimo possível na aplicação. As próximas subseções vão ao encontro de responder as sentenças-problema elencadas acima.

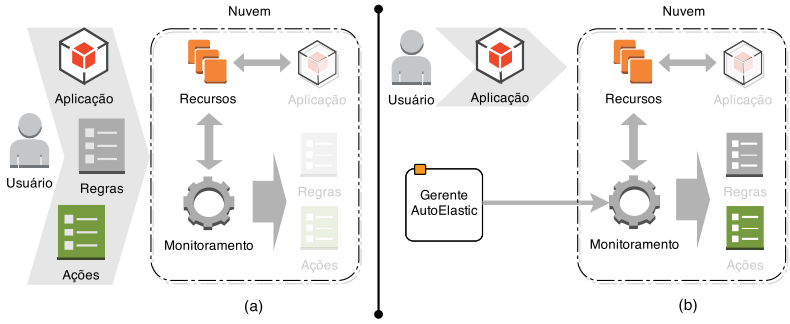


Figura 1. Uso da elasticidade: (a) Abordagem de Windows Azure e Amazon AWS na qual o usuário pré-configura regras e ações; (b) Ideia geral de AutoElastic

3.1. Arquitetura do Modelo

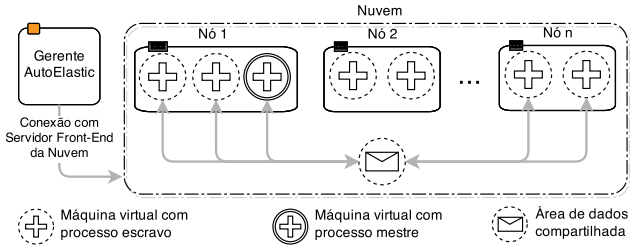


Figura 2. Arquitetura de AutoElastic sobre uma nuvem com nós que possuem dois núcleos de processamento

AutoElastic opera em nível de plataforma (PaaS) de uma nuvem computacional, atuando como um middleware que permite a transformação de uma aplicação paralela não elástica em outra elástica. Em adição, o modelo trabalha com elasticidade automática e reativa tanto na sua modalidade horizontal quanto vertical, proporcionando a alocação e

consolidação de nós computacionais e máquinas virtuais. A Figura 2 ilustra a arquitetura de componentes de AutoElastic e o mapeamento de VMs. O arcabouço contempla um Gerente, que pode ser mapeado para uma VM dentro da nuvem ou atuar como um programa fora dela. Essa flexibilidade é atingida através do uso da API (interface de programação) do middleware de nuvem a ser usado. Uma vez que normalmente aplicações de alto desempenho são intensivas quanto ao uso de CPU, optou-se por criar um processo por VM e n VMs por nó, sendo n o número de núcleos de processamento que o nó possui. Essa abordagem está baseada no trabalho de Lee et al [Lee et al. 2011], no qual busca-se explorar uma melhor eficiência em aplicações paralelas.

O Gerente AutoElastic monitora as VMs em execução e toma as decisões de elasticidade. O usuário pode informar um arquivo de SLA com o mínimo e o máximo de VMs para a execução de sua aplicação. Caso esse arquivo não seja fornecido, assume-se que o número máximo permitido de VMs é o dobro daquele informado no lançamento. O fato do Gerente, e não da aplicação em si, aumentar ou diminuir os recursos, traz o benefício da elasticidade assíncrona uma vez que a aplicação não é penalizada pela sobrecarga da (des)alocação de recursos. Entretanto, esse assincronismo acarreta na seguinte pergunta: Como avisar a aplicação da reconfiguração de recursos? Para tal, foi modelada uma comunicação entre as VMs e o Gerente AutoElastic através de uma área de dados compartilhada que pode ser viabilizada, por exemplo, via NFS, middleware orientado a mensagens (como JMS ou AMQP) ou espaço de tuplas (como JavaSpaces). O uso de uma área de dados comum para interação entre instâncias é uma abordagem corriqueira em nuvens privadas [Cai et al. 2012, Milojicic et al. 2011, Wen et al. 2012]. AutoElastic usa essa estratégia para disparar ações da seguinte forma:

- Escrita pelo Gerente AutoElastic e leitura pelos processos
  - Ação1: Há um novo recurso com n máquinas virtuais, cada qual com um novo processo da aplicação.
  - Ação2: Requisitar permissão para consolidar um nó e suas VMs.
- Escrita pelo processo mestre para posterior leitura pelo Gerente AutoElastic
  - Ação3: Dar permissão para consolidar o nó previamente requerido.

Com base na Ação 1, os processos da aplicação podem começar a trabalhar com os novos criados nos novos recursos. A Ação 2 é pertinente pelos seguintes motivos: (i) não acabar com a execução de um processo no meio de uma computação; (ii) garantir que a aplicação não seja abortada com a interrupção repentina de um dos processos. Em particular, o segundo motivo é peculiar em aplicações do tipo MPI que executam sobre TCP/IP, que normalmente são interrompidas no término prematuro de quaisquer processos. A Ação 3 é tomada pelo processo mestre, que garante que a aplicação possui um estado global consistente em que processos podem ser desconectados. A partir daí, o mestre já não passa nenhuma tarefa para o nó informado. A abordagem da área compartilhada foi adotada para que todos os processos possam saber da reconfiguração de recursos e possam estabelecer conexões entre si para troca de dados.

A técnica de elasticidade usada por AutoElastic é a de replicação [Kouki et al. 2014]. Na atividade de aumento da infraestrutura, o Gerente aloca um novo nó e lança neles novas máquinas virtuais através de um *template* vinculado a aplicação. O *boot* de uma VM é finalizado com a execução do processo escravo, que requisita comunicação com o mestre. As ações elencadas anteriormente e a replicação viabilizam a elasticidade assíncrona, uma vez que a execução da aplicação

ocorre concomitante com a reconfiguração de recursos. A instanciação é feita pelo Gerente AutoElastic e somente depois que elas estão executando, avisa-se os processos via região de dados compartilhada. Quanto a consolidação, o grão de trabalho é sempre um nó e não uma VM. Isso se deve ao fato do uso eficiente dos recursos (não usar parcialmente os núcleos disponíveis) e melhor gerência no consumo de energia elétrica. Em especial, Baliga et al. [Baliga et al. 2011] afirmam que o número de VMs em um nó não é um fator tão influente para consumo energético, e sim se o nó está ligado ou não.

Assim como em [Imai et al. 2012, Chiu and Agrawal 2010], o monitoramento do gerente para as ações de elasticidade é dado de forma periódica. De tempos em tempos é realizada a captura da métrica CPU para análise temporal e comparação com os *thresholds* máximo e mínimo. As ações de elasticidade são disparadas em situações nas quais algum dos *thresholds* é violado aplicando-se a ideia de média móvel sob um determinado número de observações. Para tal, AutoElastic coleta dados de CPU das VMs com base na função PC (Predição de Carga), conforme as Equações 1 e 2. Nesse contexto, $MM(i, j)$ é responsável por informar a carga da máquina virtual j na observação i. Para tal, MM faz uso de média móvel levando em consideraçãs as x últimas observações de carga C a partir de i. Usando esse valor, faz-se a média aritmética e estabelece-se a carga média do sistema na observação i pela função $PC(i)$, na qual n representa o número de máquinas virtuais em execução. Por fim, a Ação 1 é disparada caso PC for maior que o *threshold* máximo, enquanto que a Ação 2 é lançada quando PC for menor que o *threshold* mínimo.

$$MM(i,j) = \frac{\sum_{k=i-x+1}^{i} Cjk}{x} \;\; emque \;\; i \geq x \tag{1}$$

$$PC(i) = \frac{\sum_{j=1}^{n} MM(i,j)}{n} \tag{2}$$

3.2. Modelo de Aplicação Paralela

AutoElastic explora o paralelismo de dados e foi projetado para lidar com aplicações iterativas com passagem de mensagens. Em particular, a versão corrente do modelo ainda possui a restrição de operar com aplicações no estilo mestre-escravo. Mesmo tendo uma organização trivial, esse modelo é usado em vários algoritmos genéticos, técnica de Monte Carlo, transformações geométricas na computação gráfica, algoritmos de criptografia e aplicações no estilo SETI-at-home [Raveendran et al. 2011]. Entretanto, a Ação 1 permite que processos já existentes saibam os identificadores do novos e estabeleçam comunicação com eles. A composição do arcabouço de comunicação passou pela análise das interfaces tradicionais de MPI 1.0 e MPI 2.0. 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. A ideia foi desacolpar, de modo a gerar flexibilidade e favorecer o comportamento da elasticidade.

A Figura 3 (a) apresenta uma aplicação iterativa suportada por AutoElastic. O mestre possui uma série de tarefas, as captura sequencialmente e paraleliza uma a uma

para processamento nos demais processos. Essa captura de trabalhos é evidenciada no laço (*loop*) externo da Figura 3 (a). AutoElastic trabalha com as seguintes diretivas baseadas na Interface de MPI 2.0: (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.

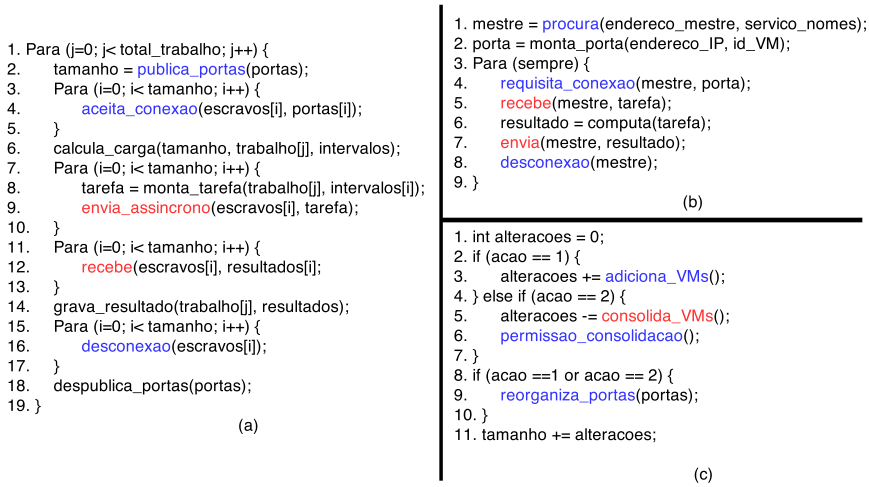


Figura 3. Modelo de aplicação: organização do processo mestre (a) e dos escravos (b); (c) código para a elasticidade, que pode substituir a linha 2 do mestre.

Referente ao código do mestre, o método da linha 2 na Figura 3 (a) verifica um arquivo de configuração ou argumentos passados para o programa que informa identificadores de máquinas virtuais e enderecos IP de cada um dos processos. Com base nisso, o mestre sabe a quantidade de escravos e cria nomes de porta para receber conexões específicas de cada um deles. Quanto à comunicação, ela acontece de forma assíncrona no processo mestre, no qual o envio de dados para os escravos é de forma não bloqueante e a recepção é bloqueante. O fato de assumir programas com um laço externo é conveniente para a elasticidade, pois logo no início dele é possível que a quantidade de recursos e processos seja reconfigurada sem alterar a semântica da aplicação. Ainda, o início de um novo laço representa um estado global consistente para o sistema distribuído.

A transformação da aplicação mostrada na Figura 3 (a) em outra elástica pode ser feita em nível PaaS através de uma das seguintes maneiras: (i) numa implementação orientada a objetos, usar polimorfismo para sobrescrever método para gerir a elasticidade; (ii) fazer um tradutor fonte-para-fonte que insira um código entre as linhas 1 e 2; (iii) desenvolvimento de um *wrapper* em linguagens procedurais para a função da linha 2 na Figura 3 (a). Independente da técnica, o código para gerenciar a elasticidade é simples e mostrado na Figura 3 (c). Primeiramente, a região de código adicional verifica no diretório compartilhado se há alguma ação nova de AutoElastic. Na ocorrência de Ação 1, o mestre lê os dados e os adiciona na região de memória de processos escravos. Se ocorrer a Ação2, o mestre retira os processos envolvidos da lista de processos e aciona a Ação3.

4. Implementação

Um protótipo foi implementado usando o sistema OpenNebula, para viabilização do ambiente de nuvem, e Java, como linguagem para a escrita da aplicação paralela e do Gerente AutoElastic. Foram criados *templates* de duas máquinas virtuais: uma para o processo mestre e outra para cada um dos escravos. O Gerente AutoElastic utiliza a própria API Java de OpenNebula para as atividades de monitoramento e gestão da elasticidades horizontal e vertical. Ainda, essa API é usada por ele para lançar a aplicação paralela na nuvem, a qual é associada a um SLA que pode ser fornecido pelo usuário. O SLA segue o padrão XML de WS-Agreement[2] e informa a quantidade mínima e máxima de MVs para teste da aplicação. O compartilhamento de dados foi implementado com NFS (Network File System). Tecnicamente, AutoElastic usa SSH para se conectar a máquina *front-end* da nuvem e a partir dali tem acesso ao diretório compartilhado NFS. Quanto a noção de carga, em nível de protótipo, a carga PC para a observação atual i de monitoramento, denominada $PC(i)$ na fórmula apresentada é dada pela média móvel da carga de todas as CPUs em execução, considerando uma janela de 3 observações. O monitoramento pelo Gerente AutoElastic, por sua vez, é periódico com o valor de 30 segundos para o intervalo de medições de desempenho. Por fim, com base em trabalhos relacionados, optou-se pela adoção de 80% para o valor máximo e 40% para o mínimo dos *thresholds* de carga.

5. Modelagem da Aplicação Paralela e Metodologia de Avaliação

A metodologia de avaliação consiste em teste de desempenho de uma aplicação científica com e sem AutoElastic, que será avaliada a luz de desempenho e uso de recursos. 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. 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, podemos escrever a integral de $f(x)$ como sendo a soma das áreas dos n trapézios contidos dentro do intervalo $[a, b]$ como mostrado na Equação 3. A Equação 4 mostra o desenvolvimento da integral segundo o método de Newton-Cotes e será usada para paralelização.

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

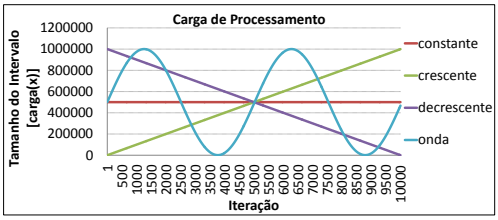
em que A_i = área do trapézio i, com $i = 0, 1, 2, 3, ..., n-1$.

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

Na Equação 4 x_0 é a e x_n é b, sendo n o número de subintervalos. Sendo assim, segundo essa equação teremos $n + 1$ equações simples no estilo $f(x)$ para calcular a intergral. Levando em consideração que o número de processos escravos seja x, então cada um deles recebe $\frac{n+1}{x}$ equações. A quantidade de subintervalos vai definir a carga de computação para cada equação que se deseja obter a integral. Para tal, foram modeladas 4 funções de carga - Crescente, Decrescente, Constante e Onda - como indica a Figura 4.

[2]http://www.ogf.org/documents/GFD.107.pdf

Por exemplo, a quantidade de subintervalos vai aumentando a cada linha na carga Crescente, enquanto que ela permanece a mesma na Constante. A Figura 4 também descreve as funções utilizadas para a obtenção dos padrões de carga.



Carga	carga(x)
Constante	1000000/2
Crescente	x * 0.2 * 500
Decrescente	1000000 - (x * 0.2 * 500)
Onda	500000 * seno(x * 0.00125) + 500500

Figura 4. Cargas de dados para processamento.

6. Avaliação e Análise dos Resultados

A aplicação que calcula a integral numérica de uma função foi avaliada com os quatro padrões de carga previamente apresentados, sob dois cenários de execução: ambos com AutoElastic, mas um deles sem habilitar a elasticidade. A Figura 5 ilustra o tempo de execução da aplicação considerando os cenários e padrões de carga abordados. Em cada execução, a configuração inicial do ambiente consistia em 2 nós, o primeiro executando 2 máquinas virtuais (2 processos escravos) e o outro 3 máquinas virtuais (2 processos escravos e 1 processo mestre). Durante a execução foram coletados dois dados: o tempo em segundos para executar a aplicação e a quantidade de observações realizadas pelo AutoElastic durante a execução. Em cada observação i realizada, tem-se a quantidade de máquinas virtuais em execução no momento, bem como $PC(i)$.

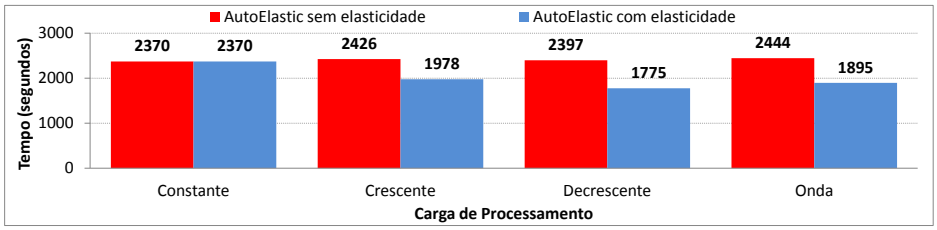


Figura 5. Tempo de execução da aplicação nos diferentes cenários e cargas.

Para avaliação, definiu-se empiricamente uma função que define o custo de execução da aplicação da seguinte forma: $custo = tempo\_total * \sum_{i=1}^{n} VMs\_SAtivas(i)$, sendo n o total de observações. A ideia é medir essa função para ambos cenários testados, com o intuito de verificar se a presença da elasticidade não possui um custo proibitivo. Ao empregar a função $custo$ para o padrão de carga Crescente com elasticidade habilitada, foram realizadas 64 observações. Deste total, foram utilizadas 4 máquinas virtuais em 31 observações, 6 máquinas virtuais em 26 observações e 8 máquinas virtuais nas 7 restantes. Aplicando a função de custo, tem-se: $(31 * 4 + 26 * 6 + 7 * 8) * 1978 = 664608$. Considerando o mesmo padrão com a elasticidade desabilitada, foram realizadas 80 observações, nas quais foram utilizadas 4 máquinas virtuais de processos escravos em todas. Aplicando a função de custo neste cenário, tem-se: $(80 * 4) * 2426 = 776320$. O padrão Decrescente obteve os valores 670950 e 708730, enquanto o padrão Onda 757452 e 791856, quando observa-se

respectivamente AutoElastic com e sem ações de elasticidade. Tais resultados mostram ganhos de 10% a 14% a favor de AutoElastic com elasticidade habiliatda.

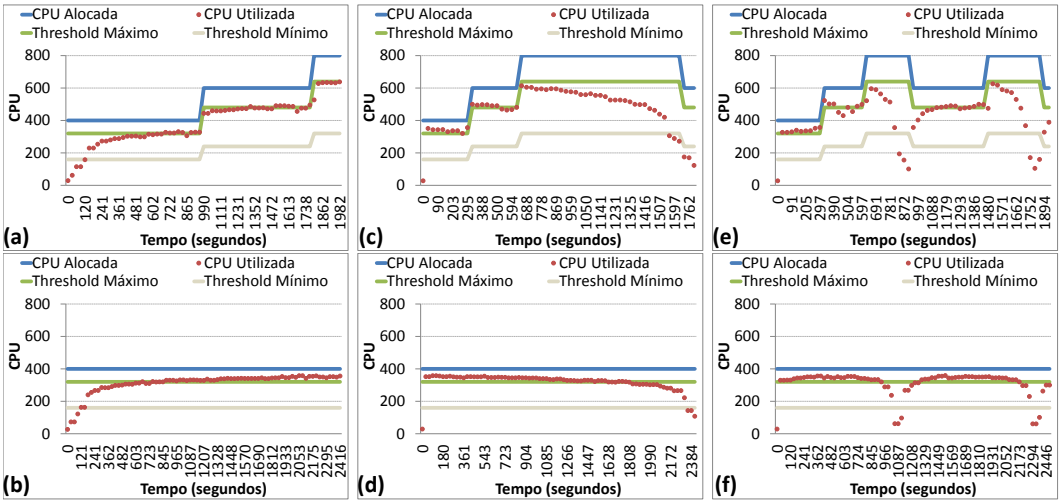


Figura 6. Uso de recursos: Cargas Crescente (a), Decrescente (c) e Onda (e) com AutoElastic e elasticidade habilitada; Cargas Crescente (b), Decrescente (d) e Onda (f) com AutoElastic, mas agora com elasticidade desabilitada.

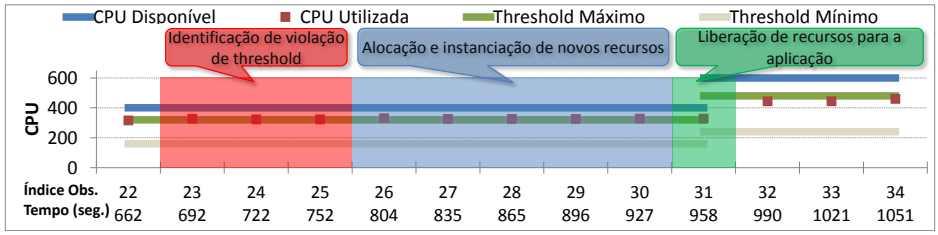


Figura 7. Identificação de violação de threshold e alocação de novos recursos

A Figura 6 mostra uma comparação do histórico de alocação de recursos quando abordados os cenários e padrões de carga. O padrão constante foi retirado pois não apresenta ações de elasticiadade, mesmo quando ela está habilitada. Como esperado, para a carga Crescente são realizadas alocações de recursos ao longo da execução, enquanto que para a carga Decrescente são alocados recursos no início e desalocados ao longo da execução. Em adição, como esperado no emprego da elasticade, o padrão Onda aloca e desaloca recursos de acordo com a flutuação de carga.

Quanto à elasticidade assíncrona, a Figura 7 demostra três momentos da execução quando uma operação de alocação é realizada: (i) análise de dados com a identificação de violação do *threshold* superior; (ii) instanciação dos recursos virtuais no ambiente e; (iii) disponibilização dos recursos para a aplicação. No ambiente de testes, a fase de instanciação compreende a transaferência de duas máquinass virtuais para o novo nó físico em uma rede de 100Mb/s e a posterior inicialização delas. Cada VM possui 700MB de tamanho para serem transferidos e o tempo total entre o seu lançamento e a sua disponibilização é de em média 214 segundos. Durante toda a fase de instanciação, a aplicação executa normalemnte com os recursos que já estavam em execução e a reorganização é feita somente depois que os novos estiverem disponíveis.

7. Conclusão

Esse artigo endereçou a elasticidade para aplicações HPC através da proposição do modelo AutoElastic. O modelo auto-organiza o número de máquinas virtuais sem a intervenção do usuário, trazendo benefícios para o administrador (melhor consumo de energia e compartilhamento de recursos entre os usuários) e para o usuário da nuvem (desempenho da aplicação e melhor gerência da sobrecarga da nuvem). A Seção 3, que descreve AutoElastic, apresentou duas sentenças-problemas, que foram abordadas da seguinte forma: (i) quanto ao uso da elasticidade, AutoElastic se destaca por atuar em nível PaaS de uma nuvem, não impondo que o programador tenha que escrever ações de elasticidade no código da aplicação. Ainda nessa linha, AutoElastic oferece elasticidade assíncrona, que se mostrou pertinente para viabilizar o uso de aplicações HPC no contexto de nuvem computacional; (ii) quanto ao modelo de aplicação, a versão corrente de AutoElastic trabalha com aplicações mestre-escravo iterativas, não precisando de informações prévias sobre o comportamento delas. Essa abordagem é justificada pelo fato que estas aplicações podem ser construídas com o estilo de programação de MPI 2.0 que segue a ideia de Sockets. Esse estilo permite que processos sejam conectados e desconectados facilmente à aplicação paralela, proporcionando um uso efetivo dos recursos disponíveis.

A avaliação demonstrou que é possível ganhar-se de 10% a 14% no tempo de execução da aplicação iterativa que calcula uma integral numérica. Em adição, os resultados ganham força quando avaliado esse desempenho juntamente com a energia consumida, mostrando que a elasticidade oferecida por AutoElastic não tem um custo proibitivo. Quanto a trabalhos futuros, pode-se citar a auto-organização dos *thresholds* de acordo com o histórico da aplicação. Por fim, planeja-se estender AutoElastic para contemplar outros modelos de programação como Divisão-e-Conquista e Fases Síncronas.

Agradecimentos

Este trabalho é parcialmente financiado pelos seguintes órgãos de pesquisa: CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico), CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) e FAPERGS (Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul).

Referências

Baliga, J., Ayre, R., Hinton, K., and Tucker, R. (2011). Green cloud computing: Balancing energy in processing, storage, and transport. *Proceedings of the IEEE*, 99(1):149–167.

Beernaert, L., Matos, M., Vilaça, R., and Oliveira, R. (2012). Automatic elasticity in openstack. In *Proceedings of the Workshop on Secure and Dependable Middleware for Cloud Monitoring and Management*, SDMCMM '12, pages 1–6, ACM.

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

Chiu, D. and Agrawal, G. (2010). Evaluating caching and storage options on the amazon web services cloud. In *Grid Computing (GRID), 11th Int. Conf. on*, pages 17 –24.

Dawoud, W., Takouna, I., and Meinel, C. (2011). Elastic vm for cloud resources provisioning optimization. *Advances in Computing and Communications*, volume 190 of *Communications in Computer and Information Science*, pages 431–445. Springer.

Goh, W. X. and Tan, K.-L. (2014). Elastic mapreduce execution. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM Int. Symp. on*, pages 216–225.

Imai, S., Chestna, T., and Varela, C. A. (2012). Elastic scalable cloud computing using application-level migration. In *Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing*, UCC '12, pages 91–98, IEEE.

Kouki, Y., Oliveira, F. A. d., Dupont, S., and Ledoux, T. (2014). A language support for cloud elasticity management. In *Cluster, Cloud and Grid Computing (CCGrid), 14th Int. Symp. on*, pages 206–215.

Lee, Y., Avizienis, R., Bishara, A., Xia, R., Lockhart, D., Batten, C., and Asanovic, K. (2011). Exploring the tradeoffs between programmability and efficiency in data-parallel accelerators. In *Comp. Arch. (ISCA), 38th Int. Symp. on*, pages 129–140.

Mao, M., Li, J., and Humphrey, M. (2010). Cloud auto-scaling with deadline and budget constraints. In *Grid Computing (GRID), 2010 11th IEEE Int. Conf. on*, pages 41 –48.

Martin, P., Brown, A., Powley, W., and Vazquez-Poletti, J. L. (2011). Autonomic management of elastic services in the cloud. In *Proc. of the 2011 IEEE Symp. on Computers and Communications*, ISCC '11, pages 135–140, IEEE.

Michon, E., Gossa, J., and Genaud, S. (2012). Free elasticity and free cpu power for scientific workloads on iaas clouds. In *Parallel and Distributed Systems (ICPADS), 18th Int. Conf. on*, pages 85 –92.

Mihailescu, M. and Teo, Y. M. (2012). The impact of user rationality in federated clouds. *Cluster Computing and the Grid, IEEE Int. Symp. on*, 0:620–627.

Milojicic, D., Llorente, I. M., and Montero, R. S. (2011). Opennebula: A cloud management tool. *Internet Computing, IEEE*, 15(2):11 –14.

Rajan, D., Canino, A., Izaguirre, J. A., and Thain, D. (2011). Converting a high performance application to an elastic cloud application. In *Proc. of the Third Int. Conf. on Cloud Computing Technology and Science*, CLOUDCOM '11, pages 383–390, IEEE.

Raveendran, A., Bicer, T., and Agrawal, G. (2011). A framework for elastic execution of existing mpi programs. In *Proc. of the 2011 IEEE Int. Symp. on Parallel and Distributed Processing Workshops and PhD Forum*, IPDPSW '11, pages 940–947, IEEE.

Roloff, E., Birck, F., Diener, M., Carissimi, A., and Navaux, P. (2012). Evaluating high performance computing on the windows azure platform. In *Cloud Computing (CLOUD), 2012 IEEE 5th Int. Conf. on*, pages 803 –810.

Sinha, N. and Khreisat, L. (2014). Cloud computing security, data, and performance issues. In *Wireless and Optical Communication Conf. (WOCC), 2014 23rd*, pages 1–6.

Wen, X., Gu, G., Li, Q., Gao, Y., and Zhang, X. (2012). Comparison of open-source cloud management platforms: Openstack and opennebula. In *Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th Int. Conf. on*, pages 2457 –2461.

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