Elasticidade Reativa em Nuvem para Aplicações de Alto Desempenho

Vinicius Facco Rodrigues[1], **Gustavo Rostirolla**[1], **Rodrigo da Rosa Righi**[1]

[1]Prog. Interdisciplinar de Pós-Graduação em Computação Aplicada, Unisinos - Brasil
Email: viniciusfacco@live.com, grostirolla1@gmail.com, rrrighi@unisinos.br

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 automatic reactive elasticity model called AutoElastic, which acts at middleware level over iterative parallel applications, offering automatic resources provisioning. 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 31% 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 reativa automática chamado AutoElastic, que atua em nível de middleware sobre aplicações paralelas iterativas. Além do modelo, o presente artigo também apresenta um protótipo construído com OpenNebula e sua avaliação com uma aplicação iterativa, demonstrando ganhos de desempenho de até 31% 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 [Kouki et al. 2014]. Apesar dos benefícios como a melhora no desempenho e a redução de custos e riscos, a elasticidade também impõe desafios para o desenvolvimento 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, 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]. 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].

Nesse contexto, esse artigo apresenta o modelo **AutoElastic**[1], que gerencia a elasticidade em aplicações HPC iterativas. AutoElastic atua em nível de middleware, 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,

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

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 recursos. Isso tem um impacto significativo no dueto HPC e elasticidade, uma vez que aplicações paralelas são sensíveis a interferências que possam piorar o desempenho. Este artigo descreve AutoElastic e um protótipo construído com o middleware OpenNebula. Testes com uma aplicação científica mostram ganhos de desempenho de até 31% quando usado AutoElastic, na comparação com provisionamento fixo.

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 [Wen et al. 2012] ou através de pré-configuração de mecanismos com elasticidade reativa [Roloff et al. 2012]. Em particular, nesse último caso, o usuário deve definir *thresholds* e ações de elasticidade, o que pode não ser trivial para usuários não especialistas.

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% [Al-Haidari et al. 2013]; (ii) 75% [Imai et al. 2012]; (iii) 80% [Al-Haidari et al. 2013]; (iv) 90% [Beernaert et al. 2012, Al-Haidari et al. 2013]. 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]; (ii) necessidade de alteração do código fonte da aplicação [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]. Neste contexto, o modelo proposto busca tratar os pontos citados e, além disso, apresentar uma análise dos diferentes *thresholds* e seu impacto no desempenho da aplicação.

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

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. Para tal, é necessário que sejam previamente definidos *thresholds* máximo e mínimo, utilizados para disparar as ações de elasticidade. Além da transparência para o usuário, há também aquela em nível de aplicação, uma vez que não são necessárias linhas de código adicionais para monitoramento ou reorganização de recursos. A transformação de uma aplicação 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) desenvolvimento de um *wrapper* em linguagens procedurais para métodos de publicação de portas.

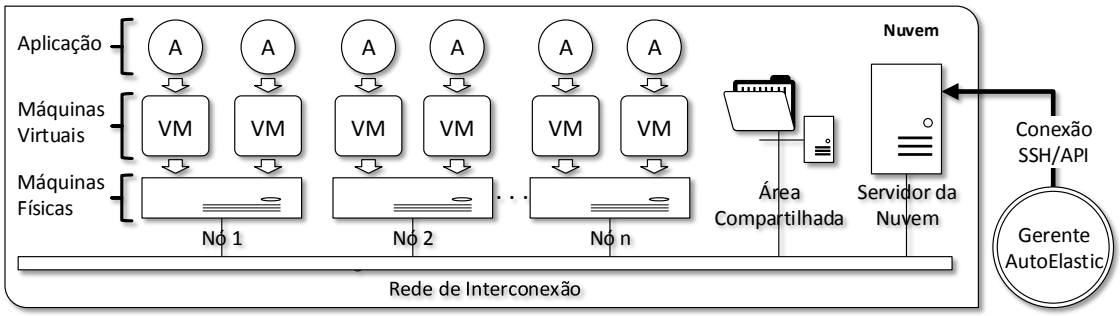

Figura 1. AutoElastic sobre uma nuvem com nós que possuem dois núcleos de processamento.

AutoElastic trabalha com elasticidade automática e reativa tanto na sua modalidade horizontal quanto vertical, proporcionando a alocação e consolidação de nós e máquinas virtuais. A Figura 1 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 cores que o nó possui. Essa abordagem está baseada em 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 realiza as operações de elasticidade quando necessário. Ao realizar alguma operação, torna-se necessário comunicar a aplicação sobre as reconfigurações do ambiente. Esta abordagem viabiliza a realização de operações de forma assíncrona, não impondo que a aplicação seja parada para as reconfigurações de recursos. Para tal, foi modelada uma comunicação entre a aplicação 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 [Wen et al. 2012]. A comunicação é realizada entre a aplicação e o Gerente AutoElastic através da escrita das seguintes ações:

- Ação 1: Gerente: Há um novo recurso com n máquinas virtuais e p processos;
- Ação 2: Gerente: Requisitar permissão para consolidar um nó e suas VMs;
- Ação 3: Aplicação: Dar permissão para consolidar o nó previamente requerido.

Com base na Ação 1, a aplicação pode começar a trabalhar com os novos processos 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. 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 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. 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.

Assim como em [Imai et al. 2012], o monitoramento do gerente para as ações de elasticidade é dado de forma periódica através da captura da métrica CPU. 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. Esta abordagem garante um melhor tratamento de picos, não ocasionando reorganizações desnecessárias de recursos. Para tal, AutoElastic coleta dados de CPU das VMs com base na função PC (Predição de Carga), conforme a equação 1. 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 consideração 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.

$$PC(i) = \frac{\sum_{j=1}^{n} MM(i,j)}{n} \qquad MM(i,j) = \frac{\sum_{k=i-x+1}^{i} Cjk}{x} \;\; em\ que \;\; i \geq x \quad (1)$$

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. O Gerente AutoElastic utiliza a própria API Java do 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 segue o padrão XML de WS-Agreement (http://www.ogf.org/documents/GFD.107.pdf), que pode ser fornecido pelo usuário definindo a quantidade máxima e mínima de nós que podem ser usados. O compartilhamento de dados foi implementado com NFS (Network File System). AutoElastic usa SSH para se conectar o servidor da nuvem e ter acesso ao diretório compartilhado NFS. Quanto a janela de observações x apresentada na equação 1, em nível de protótipo, foi adotado um valor 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.

5. Modelagem da Aplicação Paralela e Metodologia de Avaliação

A metodologia de avaliação consiste em teste de desempenho de uma aplicação científica com AutoElastic e elasticidade reativa habilitada/desabilitada. 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 $[a, b]$.

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, variando a quantidade de subintervalos em cada iteração, foram modeladas 4 funções de carga com 10000 iterações cada: Crescente, Decrescente, Constante e Onda.

Cada carga foi executada em dois cenários, ambos iniciando com 2 nós e 4 VMs: (i) AutoElastic com a elasticidade reativa habilitada e (ii) AutoElastic com a elasticidade reativa desabilitada. Para o cenário ii, todas as cargas foram executadas em 25 situações diferentes através da variação dos *thresholds* máximos e mínimos. Considerando as escolhas dos diferentes trabalhos, foram adotados para *threshold* máximo os valores 70%, 75%, 80%, 85% e 90%, enquanto que 30%, 35%, 40%, 45% e 50% para o mínimo.

Além da perspectiva de desempenho em relação ao tempo de execução da aplicação, a análise de consumo de recursos durante a execução da aplicação é realizada através da equação $C = \sum_{i=1}^{n}(i * T(i))$, em que $T(i)$ refere-se ao tempo em que a aplicação utilizou i VMs e n é a quantidade máxima de VMs utilizadas.

6. Avaliação e Análise dos Resultados

A tabela 1 apresenta as combinações de *thresholds* que obtiveram os melhores e o piores resultados para cada padrão de carga, quando analisado o tempo total de execução da aplicação. Em adição, também são apresentados os resultados obtidos pela aplicação quando executada sem a utilização da elasticidade reativa gerida por AutoElastic.

Através da tabela, é possível notar que os melhores resultados foram obtidos pelas combinações em que o *threshold* máximo era menor. Por outro lado, *thresholds* máximos maiores obtiveram os piores resultados. Esse comportamento ocorre devido ao fato de que quanto maior for o *threshold*, maior é o tempo em que os processos executam sobrecarregados. Em adição, quanto menor o *threshold* mais rápido são alocados novos recursos aumentando o poder de processamento, impactando diretamente no desempenho.

Analisando o consumo obtido em cada cenário, é possível notar que os melhores desempenhos obtiveram maiores consumos em comparação com os piores desempenhos e com os cenários sem elasticidade. Esse comportamento está ligado aos *thresholds* máximos que, quanto menores, mais recursos são alocados para a aplicação, ocasionando um maior consumo. Por outro lado, *thresholds* máximos altos diminuem a quantidade de recursos alocados devido ao fato de a aplicação demorar mais para atingir tais valores.

Elasticidade	Padrão de Carga	Melhores Resultados				Piores Resultados			
		Thresholds		Tempo	Consumo	Thresholds		Tempo	Consumo
		Máximo	Mínimo			Máximo	Mínimo		
Habilitada	Crescente	70	Todos	1601	11160	90	Todos	2354	9416
	Constante	70	Todos	1569	11206	85	Todos	2305	9220
	Decrescente	70	50	1580	12014	90	Todos	2334	9336
	Onda	70	30	1730	12518	90	Todos	2383	9532
Desabilitada	Crescente	-	-	2317	9268	-	-	-	-
	Constante	-	-	2281	9124	-	-	-	-
	Decrescente	-	-	2308	9232	-	-	-	-
	Onda	-	-	2345	9380	-	-	-	-

Tabela 1. Resultados da variação de thresholds considerando o tempo de execução da aplicação.

7. Conclusão

AutoElastic atua como um middleware que permite a transformação de uma aplicação paralela não elástica em outra elástica. Ainda, as aplicações suportadas por AutoElastic são aquelas que utilizam passagem de mensagens com paralelismo explícito. Esse estilo

permite que processos sejam conectados e desconectados facilmente à aplicação paralela, proporcionando um uso efetivo dos recursos disponíveis. Uma comparação entre a utilização da elasticidade em relação a execução da aplicação sem a elasticidade demonstrou que a variação do *threshold* máximo impacta diretamente no desempenho da aplicação. Resultados demonstraram que a aplicação obteve melhor desempenho quando utilizado um *threshold* máximo de 70% para todos os padrões de carga, chegando a 31% de ganhos no tempo de execução. Em contrapartida, *thresholds* máximos próximos a 100% não se mostram boas escolhas obtendo piores desempenhos.

Quanto a trabalhos futuros, planeja-se estender AutoElastic para contemplar outros modelos de programação como Divisão-e-Conquista e Fases Síncronas. Por fim, pode-se citar a auto-organização dos *thresholds* de acordo com o histórico da aplicação e análises de diferentes parametrizações quanto a granularidade do monitoramento.

Referências

Al-Haidari, F., Sqalli, M., and Salah, K. (2013). Impact of CPU Utilization Thresholds and Scaling Size on Autoscaling Cloud Resources. *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, pages 256–261.

Beernaert, L., Matos, M., Vilaça, R., and Oliveira, R. (2012). Automatic elasticity in openstack. In *Proc. of the Workshop on Secure and Dependable Middleware for Cloud Monitoring and Man.*, SDMCMM '12, pages 2:1–2:6, New York, NY, USA. ACM.

Imai, S., Chestna, T., and Varela, C. A. (2012). Elastic scalable cloud computing using application-level migration. In *Proc. of the 2012 IEEE/ACM Fifth Int. Conf. on Utility and Cloud C.*, UCC '12, p. 91–98, Washington, DC, USA. IEEE Computer Society.

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), 2014 14th IEEE/ACM International Symposium 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), 2011 38th An. Int. Symp. on*, p. 129–140.

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), 2012 IEEE 18th International Conference on*, pages 85 –92.

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 Par. and Dist. Proc. Wor. and PhD For.*, IPDPSW '11, p. 940–947, WA, DC, USA. IEEE Computer Society.

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 International Conference 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 International Conference on*, pages 2457 –2461.