Elasticidade Assíncrona: Transferência não Bloqueante de VMs para Viabilizar a Reorganização de Aplicações HPC em Cloud Computing

Vinicius F. Rodrigues[1], **Gustavo Rostirolla**[1], **Rodrigo da R. Righi**[1]

[1]Programa Interdisciplinar de Pós-Graduação em Computação Aplicada (PIPCA)
Universidade do Vale do Rio dos Sinos (UNISINOS)

viniciusfacco@live.com, grostirolla1@gmail.com, rrrighi@unisinos.br

Resumo. *A elasticidade é uma das principais funcionalidades de ambientes de computação em nuvem. Considerando a área de computação de alto desempenho (HPC), aplicações possuem dificuldade para usufruir deste recurso sendo necessárias alterações no código fonte da aplicação para tratar a elasticidade, impondo sobrecarga na aplicação. Neste contexto, este artigo apresenta um modelo de elasticidade assíncrona em nuvem para aplicações HPC iterativas em que as operações de elasticidade são executadas simultaneamente à execução da aplicação não impondo bloqueios ou modificações de código fonte.*

Abstract. *Elasticity is one of the main characteristics of cloud computing. Regarding the high performance computing (HPC) area, it is difficult for applications deal with dinamic environments, where it is necessary to modify the application code to treat the elasticity which imposes overhead on application for these tasks. In this context, this paper presents an asynchronous elasticity cloud model for iterative HPC applications where elasticity operations are performed simultaneously with the application execution not imposing blocking or source code modifications.*

1. Introdução

Uma das principais funcionalidades de ambientes de computação em nuvem é a elasticidade, em que usuários podem reorganizar a disponibilidade de recursos a qualquer momento de acordo com a demanda ou necessidade de desempenho [Lorido-Botran et al. 2014, Raveendran et al. 2011]. Considerando o escopo de aplicações de alto desempenho (HPC) e aplicações paralelas de longa duração, um usuário pode querer aumentar a quantidade de recursos disponíveis com o objetivo de reduzir o tempo total de execução da aplicação. Apesar de transparente para o usuário, esse tipo de mecanismo é apropriado para aplicações fracamente acopladas em que réplicas não estabelecem comunicação entre si e a distribuição de tarefas é realizada por um balanceador de carga [Galante and Bona 2012, Jennings and Stadler 2014]. Devido a isso, a elasticidade em nuvem é mais explorada em arquiteturas WEB cliente-servidor, como vídeo sob demanda, lojas online, aplicações BOINC, *e-governance* e serviços WEB [Raveendran et al. 2011]. Embora pertinente para aplicações do estilo sacola-de-tarefas, técnicas de replicação e balanceadores de carga centralizado não são funcionais por padrão para implementar elasticidade em aplicações HPC fortemente acopladas, como aquelas modeladas como *Bulk-Synchronous Parallel* (BSP), divisão-e-conquista ou pipeline [Raveendran et al. 2011, Frincu et al. 2013].

Grande parte das aplicações paralelas são desenvolvidas utilizando *Message Passing Interface* (MPI) 1, em que não há suporte para alteração da quantidade de processos durante a execução [Wilkinson and Allen 2005]. Por outro lado, com a segunda versão de MPI, em que há suporte para a dinamicidade de processos, é necessário um esforço significante em nível de aplicação para manualmente alterar o grupo de processos e redistribuir os dados para efetivamente utilizar uma quantidade de processos diferente e tirar proveito da elasticidade. Abordagens que requerem a reescrita do código impõem sobrecarga na aplicação, a qual, além de realizar tarefas de monitoramento, deve coordenar a reorganização de recursos. Neste contexto, com o objetivo de oferecer elasticidade em nuvem para aplicações HPC de maneira eficiente e transparente, é proposto um modelo de elasticidade assíncrona viabilizado através do modelo de elasticidade AutoElastic [Righi et al. 2015]. A contribuição de AutoElastic conta com o conceito de elasticidade assíncrona: reorganização transparente de recursos e processos pela perspectiva do usuário, não bloqueando ou finalizando a execução da aplicação em qualquer ação de alocação ou desalocação de recursos. Para realizar isso, AutoElastic oferece um *framework* com um controlador que transparentemente gerencia ações de elasticidade horizontal, não sendo necessárias modificações na aplicação.

2. Trabalhos Relacionados

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* [Raveendran et al. 2011]. Tal ação pode ter um impacto negativo, especialmente para aplicações HPC que não têm longa duração. Em adição, a abordagem de ElasticMPI necessita alteração no código fonte da aplicação a fim de inserir políticas de monitoramento. Mao, Li e Humphrey [Mao et al. 2010] 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. [Martin et al. 2011] 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, Elastack aparece como um sistema executando sobre OpenStack para suprir sua falta de elasticidade [Beernaert et al. 2012].

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* [Martin et al. 2011, Beernaert et al. 2012]; (ii) necessidade de alteração no código fonte da aplicação [Raveendran et al. 2011, Rajan et al. 2011]; (iii) necessidade de conhecer previamente dados da aplicação antes de sua execução, como tempo de execução esperado para cada componente [Raveendran et al. 2011]; e (iv) necessidade de reconfigurar recursos com a parada da aplicação e subsequente recuperação [Raveendran et al. 2011]. Observando os trabalhos mencionados, é possível destacar três deles que focam elasticidade em nuvem para aplicações HPC [Raveendran et al. 2011, Martin et al. 2011, Rajan et al. 2011]. Eles têm em comum a abordagem do modelo de programação mestre-escravo. Particularmente, as iniciativas [Raveendran et al. 2011] e [Rajan et al. 2011] 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 [Rajan et al. 2011] 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. [Martin et al. 2011] é o tratamento eficaz das requisições de um servidor Web através da criação e consolidação de instâncias baseando-se no fluxo de requisições e carga das VMs trabalhadoras.

3. Modelo de Elasticidade Assíncrona

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 baseada em *thresholds* de forma horizontal, proporcionando alocação e consolidação de nós computacionais e máquinas virtuais. Como uma proposta PaaS, AutoElastic propõe um *middleware* para compilar uma aplicação mestre-escravo iterativa, além de um gerenciador de elasticidade. O Gerenciador esconde do usuário os detalhes da escrita de regras e ações de elasticidade. A Figura 1 apresenta a arquitetura do modelo, ilustrando a relação entre os processos, máquinas virtuais e nós computacionais. Levando em consideração que aplicações HPC são comumente intensivas quanto a CPU, optou-se pela criação de um único processo por VM e c VMs por nó de computação buscando explorar uma melhor eficiência em aplicações paralelas. Dessa maneira, uma nuvem AutoElastic é modelada com m nós homogêneos, os quais possuem c máquinas virtuais cada, em que c é a quantidade de núcleos de processamento de cada nó. Por fim, a qualquer momento, o número de VMs executando processos escravos é igual a $n = c \times m$.

O fato de que o Gerenciador AutoElastic, e não a aplicação por si, realiza as operações de reconfiguração de recursos oferece o benefício da elasticidade assíncrona. Elasticidade assíncrona é uma maneira de assincronamente notificar uma aplicação que está executando em nuvem sobre mudanças na disponibilidade de recursos do ambiente, tais como o número de instâncias de máquinas virtuais. Por exemplo, a aplicação é notificada assim que uma nova instância de uma máquina virtual está disponível no ambiente, sem prejudicar seu fluxo de execução normal. No entanto, esta operação não-

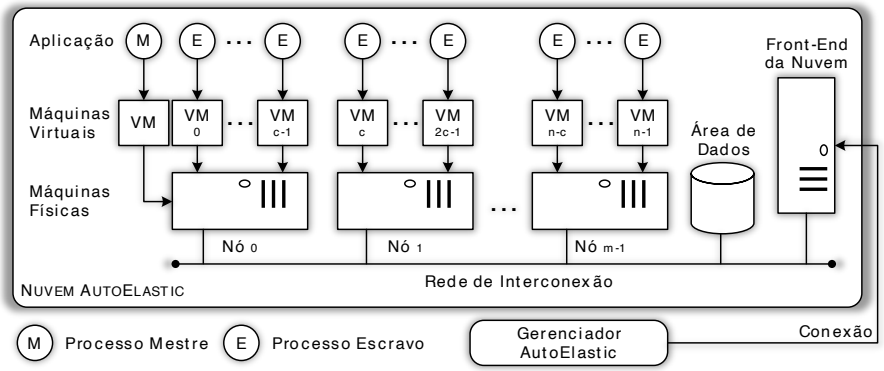

Figura 1. Distribuição de nós, VMs e processos em uma infraestrutura de nuvem AutoElastic, na qual cada VM engloba um único processo da aplicação e cada nó executa c VMs, em que c denota o número de núcleos de processamento do nó.

bloqueante implica na seguinte pergunta: *Como podemos notificar a aplicação sobre a reconfiguração de recursos?* Para isso, AutoElastic oferece a comunicação entre as máquinas virtuais e o Gerenciador AutoElastic utilizando uma área de memória compartilhada. Essa é uma área privada para as máquinas virtuais e as máquinas físicas dentro do ambiente da nuvem AutoElastic. A comunicação entre as máquinas virtuais e o Gerenciador AutoElastic através dessa área pode ser viabilizada, por exemplo, via NFS, *middleware* orientado a mensagens (tais como JMS ou AMQP) ou espaço de tuplas (como JavaSpaces). O uso de uma área compartilhada para interação de dados entre máquinas virtuais é uma abordagem comum em nuvens privadas [Cai et al. 2012].

A Tabela 1 apresenta as Notificações suportadas por AutoElastic. Baseado na Notificação 1, 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 Notificação 2 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. A Notificação 3 é tomada pelo processo mestre, que garante que a aplicação está em um estado global consistente em que processos podem ser desconectados apropriadamente.

Tabela 1. Notificações fornecidas através da área de dados compartilhada.

Notificação	Direção	Descrição
Notificação 1	Gerenciador AutoElastic → Processo Mestre	Disponível c novos recursos.
Notificação 2	Gerenciador AutoElastic → Processo Mestre	Solicitação de permissão para consolidação de recursos.
Notificação 3	Processo Mestre → Gerenciador AutoElastic	Resposta à Notificação 2 permitindo a consolidação.

Como em Imai et al. [Imai et al. 2012], o monitoramento de dados é realizado de forma periódica. Assim, o Gerente AutoElastic 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 a Equação 1. $PC(i)$ retorna uma previsão de carga de CPU quando considerando a execução de n VMs com processos escravos 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. $MM(i, j)$ calcula a 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 Notificação 1 é disparada se PC for maior que o *threshold* superior, enquanto a Notificação 2 é acionada quando PC for menor que o *threshold* inferior.

$$PC(i) = \frac{1}{n} \times \sum_{j=0}^{n-1} MM(i,j) \qquad MM(i,j) = \frac{\sum_{k=i-z+1}^{i} Carga(k,j)}{z} \tag{1}$$

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. O modelo proposto atua segundo a 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.

4. Metodologia de Avaliação

Foi configurado uma nuvem privada, utilizando o ambiente OpenNebula 4.2 em um *cluster* com 10 nós homogêneos, para a execução de uma aplicação que 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 [Comanescu 2012]. Considere a partição do intervalo $[a, b]$ em s subintervalos iguais, cada qual de comprimento h ($[x_i, x_{i+1}]$, para $i = 0, 1, 2, ..., s - 1$). Assim, $x_{i+1} - x_i = h = \frac{b-a}{s}$. Dessa forma, pode-se escrever a integral de $f(x)$ como sendo a soma das áreas dos s trapézios contidos dentro do intervalo $[a, b]$. 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. Como s define a quantidade de sub-intervalos, e consequentemente a quantidade de equações, seu valor define a carga computacional necessária para atingir o resultado final para uma equação em particular. Utilizando esse parâmetro, foram modelados quatro padrões de carga conforme demonstrado na Figura 2.

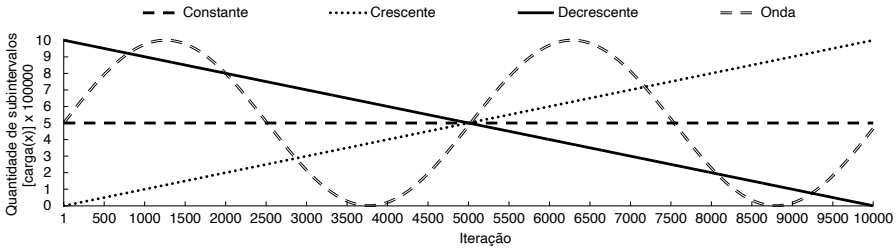

Figura 2. Modelos de padrões de carga. O eixo x expressa a iteração (cada iteração representa uma equação que será calculada), enquanto o eixo y representa a respectiva carga de processamento para aquela iteração (valor de s).

A aplicação foi executada com cada padrão de carga em diferentes cenários com a elasticidade habilitada, variando a configuração de *thresholds*. Assumindo as escolhas de diferentes trabalhos, nos quais podem ser encontrados *thresholds* superiores como 50% [Al-Haidari et al. 2013], 70% [Dawoud et al. 2011, Al-Haidari et al. 2013], 75% [Imai et al. 2012], 80% [Suleiman 2012, Al-Haidari et al. 2013] e 90% [Beernaert et al. 2012, Al-Haidari et al. 2013], foram adotados 70%, 75%, 80%, 85% e 90%, enquanto que para os *thresholds* inferiores foram adotados 30%, 35%, 40%, 45% e 50% baseando-se no trabalho de Haidari et al. [Al-Haidari et al. 2013].

5. Análise de Resultados

A aplicação foi executada com cada padrão de carga em diferentes configurações de *thresholds*, resultando em diferentes necessidades de reconfiguração de recursos. Durante a realização dos testes, foram realizadas 20 operações para a carga Constante, 50 para

a carga Crescente, 40 para a carga Decrescente e 83 para a carga Onda, totalizando 193 operações de alocação de recursos. Para cada operação, foi obtido o tempo total entre o momento em que os recursos são alocados no ambiente e o momento em que os novos recursos são disponibilizados para o uso da aplicação. Das 193 operações, o tempo médio obtido foi de 152,71 segundos, com uma mediana 148 segundos e um desvio padrão de 6,99 segundos. Durante todo o tempo de execução destas operações, a aplicação continuou executando simultaneamente sem ser afetada pelas tarefas relacionadas à alocação de recursos. A Tabela 2 apresenta o tempo total de execução da aplicação para cada uma das execuções, além de apresentar a quantidade total de operações de alocação de recursos que foram realizadas em cada execução e o tempo total dessas operações.

Tabela 2. Resultados obtidos em todos os cenários com todos os padrões de carga. Tempos estão em segundos e o valor entre parênteses do campo Tempo Alocação representa a quantidade de alocações realizadas.

Thresholds		Constante		Crescente		Decrescente		Onda	
Superior	Inferior	Tempo Execução	Tempo Alocação	Tempo Execução	Tempo Alocação	Tempo Execução	Tempo Alocação	Tempo Execução	Tempo Alocação
70	30	1569	310 (2)	1578	602 (4)	1602	458 (3)	1730	759 (5)
	35	1569	310 (2)	1578	602 (4)	1609	457 (3)	1733	753 (5)
	40	1569	310 (2)	1578	602 (4)	1606	448 (3)	1742	740 (5)
	45	1569	310 (2)	1578	602 (4)	1596	458 (3)	1829	886 (6)
	50	1569	310 (2)	1578	602 (4)	1580	441 (3)	1835	904 (6)
75	30	1799	162 (1)	1776	459 (3)	1689	327 (2)	1755	752 (5)
	35	1799	162 (1)	1776	459 (3)	1684	295 (2)	1854	756 (5)
	40	1799	162 (1)	1776	459 (3)	1680	311 (2)	1751	754 (5)
	45	1799	162 (1)	1776	459 (3)	1675	312 (2)	1864	737 (5)
	50	1799	162 (1)	1776	459 (3)	1659	310 (2)	1853	900 (6)
80	30	1781	162 (1)	1933	309 (2)	1679	310 (2)	1907	591 (4)
	35	1781	162 (1)	1933	309 (2)	1680	310 (2)	1877	604 (4)
	40	1781	162 (1)	1933	309 (2)	1675	310 (2)	1892	606 (4)
	45	1781	162 (1)	1933	309 (2)	1673	312 (2)	1921	622 (4)
	50	1781	162 (1)	1933	309 (2)	1662	312 (2)	1891	594 (4)
85	30	2305	0	2137	162 (1)	1872	164 (1)	2118	310 (2)
	35	2305	0	2137	162 (1)	1860	162 (1)	2097	310 (2)
	40	2305	0	2137	162 (1)	1857	163 (1)	2080	296 (2)
	45	2305	0	2137	162 (1)	1859	162 (1)	2031	294 (2)
	50	2305	0	2137	162 (1)	1850	146 (1)	2082	308 (2)
90	TODOS	2297	0	2348	0	2334	0	2383	0

Através da Tabela 2 é possível identificar que quanto menor é o valor para o *threshold* superior, maior é a quantidade de operações de alocação necessárias. Isso se deve ao fato de que, com um valor baixo para esse *threshold*, a carga de processamento atinge o nível do *threshold* de maneira mais rápida, resultando em um número maior de operações necessárias. Além disso, a execução da aplicação com a carga Onda resultou em um maior número de operações devido ao seu comportamento variável em que são desalocados recursos em momentos de carga baixa e alocados novamente quando a carga de processamento volta a crescer. Uma maior quantidade de alocações resulta em maior tempo em que recursos estão sendo alocados em simultâneo com a aplicação. Graças a elasticidade assíncrona, esses tempos são escondidos da aplicação que não sofre nenhum bloqueio enquanto as operações estão ocorrendo. As execuções com o *threshold* superior 70 foram responsáveis pela maior parte das operações de alocação de recursos totalizando 73 das 193 operações. Considerando essas execuções com o *threshold* superior 70, para a carga Constante o tempo de alocação equivaleu em média a 19,76% do tempo total de execução da aplicação. Já para a carga Crescente, esta equivalência atingiu o valor médio de 38,15% enquanto que para a carga Decrescente esse valor médio foi de 28,30%. Os

maiores valores obtidos foram pela carga Onda em que o tempo de alocação de recursos equivaleu em média a 45,5% do tempo total de execução da aplicação.

Por fim, em todas as execuções com o *threshold* superior 90 não foram realizadas operações pois o valor de 90% não foi atingido em nenhum momento. Devido a isso, os tempos de execução com esse *threshold* equivalem ao tempo de execução da aplicação sem elasticidade. Com isso, é possível notar que o desempenho das execuções em que houveram alocações é melhor em comparação com a execução sem elasticidade.

6. Conclusão

Este artigo apresentou um modelo de elasticidade assíncrona horizontal em nuvem para aplicações HPC iterativas chamado AutoElastic. Apesar de que Spinner et al. [Spinner et al. 2014] afirmarem que apenas elasticidade vertical é adequada para cenários HPC devido a sobrecarga nas operações de reorganização de recursos, com a elasticidade assíncrona, torna-se viável a utilização da elasticidade horizontal sem impactar na execução da aplicação. Dessa maneira, a execução da aplicação e as ações de elasticidade ocorrem simultaneamente, não penalizando a aplicação com a sobrecarga da reconfiguração de recursos. AutoElastic é ciente quanto à sobrecarga para instanciar uma máquina virtual, escondendo da aplicação essa penalização. Ainda, AutoElastic conta com um *middleware* e um gerenciador não havendo necessidade de modificações no código fonte da aplicação para tratar o ambiente dinâmico. Resultados demonstraram que em alguns cenários em que muitas operações de elasticidade são executadas, o tempo total de alocação de recursos pode chegar a 49,26% do tempo total de execução da aplicação, dependendo do quão longa é a execução da aplicação.

Futuramente, planeja-se estender o trabalho para diferentes modelos de elasticidade com o uso de *thresholds* dinâmicos, além de abordar diferentes modelos de programação como BSP e pipeline.

Referências

Al-Haidari, F., Sqalli, M., and Salah, K. (2013). Impact of cpu utilization thresholds and scaling size on autoscaling cloud resources. In *Cloud Comp. Technology and Science (CloudCom), 2013 IEEE 5th Int. Conf. on*, volume 2, 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 Manag.*, SDMCMM '12, pages 2:1–2:6, New York, NY, USA. 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 Int. Conf. on*, pages 400 –404.

Comanescu, M. (2012). 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 Int. Conf. on*, pages 1–6.

Dawoud, W., Takouna, I., and Meinel, C. (2011). Elastic vm for cloud resources provisioning optimization. In Abraham, A., Lloret Mauri, J., Buford, J., Suzuki, J., and Thampi, S., editors, *Advances in Comp. and Communications*, volume 190 of *Communications in Computer and Information Science*, pages 431–445. Springer Berlin Heidelberg.

Frincu, M. E., Genaud, S., and Gossa, J. (2013). Comparing provisioning and scheduling strategies for workflows on clouds. In *Proc. of the 2013 IEEE 27th Int. Symposium on Parallel and Distributed Processing Workshops and PhD Forum*, IPDPSW '13, pages 2101–2110, Washington, DC, USA. IEEE Computer Society.

Galante, G. and Bona, L. C. E. d. (2012). A survey on cloud computing elasticity. In *Proc. of the 2012 IEEE/ACM Fifth Int. Conf. on Utility and Cloud Comp.*, UCC '12, pages 263–270, Washington, DC, USA. IEEE Computer Society.

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 Comp.*, UCC '12, pages 91–98, Washington, DC, USA. IEEE Computer Society.

Jennings, B. and Stadler, R. (2014). Resource management in clouds: Survey and research challenges. *Journal of Network and Sys. Manag.*, pages 1–53.

Lorido-Botran, T., Miguel-Alonso, J., and Lozano, J. (2014). A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of Grid Comp.*, 12(4):559–592.

Mao, M., Li, J., and Humphrey, M. (2010). Cloud auto-scaling with deadline and budget constraints. In *Grid Comp. (GRID), 2010 11th IEEE/ACM 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 Symposium on Computers and Communications*, ISCC '11, pages 135–140, Washington, DC, USA. IEEE Computer Society.

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 2011 IEEE Third Int. Conf. on Cloud Comp. Technology and Science*, CLOUDCOM '11, pages 383–390, Washington, DC, USA. IEEE Computer Society.

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

Righi, R., Rodrigues, V., Andre daCosta, C., Galante, G., Bona, L., and Ferreto, T. (2015). Autoelastic: Automatic resource elasticity for high performance applications in the cloud. *Cloud Comp., IEEE Transactions on*, PP(99):1–1.

Spinner, S., Kounev, S., Zhu, X., Lu, L., Uysal, M., Holler, A., and Griffith, R. (2014). Runtime Vertical Scaling of Virtualized Applications via Online Model Estimation. In *Proc. of the 2014 IEEE 8th Int. Conf. on Self-Adaptive and Self-Organ. Sys. (SASO)*.

Suleiman, B. (2012). Elasticity economics of cloud-based applications. In *Proc. of the 2012 IEEE Ninth Int. Conf. on Services Comp.*, SCC '12, pages 694–695, Washington, DC, USA. IEEE Computer Society.

Wilkinson, B. and Allen, C. (2005). *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*. An Alan R. Apt book. Pearson/Prentice Hall.