Elasticidade Automática de Recursos para Aplicações de Alto Desempenho em Ambientes de Computação em Nuvem

Vinicius Facco Rodrigues
Universidade do Vale do Rio dos Sinos
CEP: 93.022–000 – São Leopoldo – Brazil
Email: viniciusfacco@live.com

Cristiano André da Costa, Rodrigo da Rosa Righi
PPG em Computação Aplicada (PIPCA)
Universidade do Vale do Rio dos Sinos
CEP: 93.022–000 – São Leopoldo – Brazil
Email: {cac,rrrighi}@unisinos.br

***Resumo*—A elasticidade é sem dúvida uma das características mais marcantes da computação em nuvem que pode ser usada na execução de aplicações que demandam processamento de alto desempenho (HPC). Normalmente, a elasticidade é oferecida em sistemas Web com o aumento e a diminuição de máquinas virtuais (MVs) que tratam requisições. Para HPC, a maioria dos sistemas necessita alterar a aplicação ou partir de um conhecimento prévio da aplicação para o tratamento dessa característica da nuvem. Nesse contexto, esse artigo apresenta o modelo de elasticidade na camada de Plataforma como Serviço (PaaS) chamado AutoElastic. Seu diferencial está no tratamento automático e sem esforço da elasticidade em aplicações de alto desempenho sem a intervenção do usuário e sem alterações no código da aplicação. A avaliação do protótipo sobre o sistema OpenNebula mostrou um ganho de desempenho de 14% no tempo de execução de uma aplicação e uma baixa intrusividade de AutoElastic na execução da aplicação paralela.**

I. Introdução

Algumas das características mais marcantes que distinguem a computação em nuvem de outras abordagens de sistemas distribuídos é a elasticidade [1]. 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 dinamicamente de acordo com a demanda do serviço ou do usuário. Nesse sentido, ela é um princípio essencial para o modelo de nuvem, pois não fornece somente o compartilhamento eficiente de recursos entre usuários, mas também é pertinente para a viabilização de computação no estilo *pay-as-you-go*.

Múltiplos provedores de computação em nuvem estão focando em aplicações que demandam alto desempenho (HPC). Aplicações de HPC na sua maioria possuem dificuldade para usufruir da elasticidade visto que normalmente são construídas com um número fixo de processos. 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. A elasticidade nesse contexto possui algumas peculiaridades. Um ou mais processos da aplicação deve proativamente ou de forma periódica analisar se há novos recursos para assim utilizá-lo e proceder com o mecanismo de balanceamento de carga. Por exemplo, em 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 [2].

Nesse contexto, esse artigo apresenta um modelo chamado AutoElastic que gerencia a elasticidade em aplicações HPC. Ele provê dinamicidade na alocação dos recursos sem a intervenção do programador. O gerenciador proposto se destaca por operar sem conhecimento prévio da aplicação, desconhecendo por exemplo o tempo esperado de suas fases. Em termos de contribuições científicas, é possível destacar os seguintes pontos: (i) gerenciamento eficiente da elasticidade com o intuito de evitar a alocação e desalocação de MVs de forma desnecessária (*trashing*) baseada na técnica de Envelhecimento (*Aging*) [3]; (ii) assincronismo na criação e destruição de MVs, de modo que a aplicação não espere de forma bloqueada pela conclusão desses procedimentos. Esse artigo descreve em detalhes a implementação de um protótipo de AutoElastic que usa o *middleware* OpenNebula. Os testes mostram resultados encorajadores e ganhos de desempenho de até 14% quando comparadas a solução proposta e a alocação estática de recursos.

II. AutoElastic: Gerenciador de Elasticidade

Uma das primeiras medidas para o desenvolvimento de AutoElastic é a definição de regras de elasticidade. Tal tarefa não é trivial por vários motivos. Primeiro, ela requer que sejam configuradas regras que usam um ou mais *thresholds* para uso dos recursos (por exemplo, uso de CPU e memória), a quantidade de ocorrência, a janela de monitoramento e eventuais adaptações em circunstância da aplicação. Segundo, regras otimizadas para o comportamento de uma aplicação podem apresentar um desempenho ruim para outra. Por fim, a elasticidade muitas vezes acarreta na alocação de recursos que por sua vez impactam diretamente no custo de uso da nuvem.

A principal ideia de AutoElastic é oferecer um modelo que possa ser usado para criar aplicações elásticas, que estejam aptas para se adaptar automaticamente de acordo com mudanças no número de recursos (em ambos os níveis nó computacional e máquina virtual). AutoElastic deve tratar com aplicações que não são regidas por *deadlines* e que podem apresentar irregularidades de computação ao longo de sua execução. Em adição, o foco é aplicações com passagem de mensagens e paralelismo explícito, com uso de diretivas de envio e recebimento de dados direto no código fonte. Ainda, AutoElastic deve ser ciente do tempo de lançamento de uma MV e deve trabalhar de modo que essa sobrecarga impacte o mínimo possível na execução da aplicação.

A Figura 1 ilustra a atuação de AutoElastic, onde a elasticidade é feita de maneira reativa e sem a necessidade de preconfiguração de métricas pelo usuário. Abaixo estão listadas as decisões de projeto, bem como as condições para o funcionamento de AutoElastic.

- Usuário não precisa preconfigurar tampouco escrever regras e ações. Em contrapartida, ele deve apresentar acordo de nível de serviço (SLA) que informa o número mínimo e máximo de Nós que podem ser alocadas.
- A aplicação paralela não precisa ser reescrita para ter o caráter elástico.
- O cenário investigado por AutoElastic assume um ambiente que não é compartilhado por outros usuários. Ele contempla um usuário que executa um único programa.
- AutoElastic oferece uma elasticidade reativa, automática, em ambas as modalidades horizontal e vertical [2], e segue o método de replicação.
- O nível de atuação é o PaaS, que contempla uma ferramenta que transforma a aplicação paralela em outra elástica de forma transparente para o usuário.
- Análise de picos de carga e quedas bruscas para não lançar ações de elasticidade de forma desnecessária, minimizando assim um fenômeno conhecido como *thrashing*.

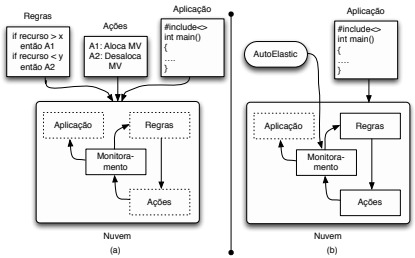

Figura 1. (a) Abordagem usada no Windows Azure e Amazon AWS na qual o usuário pré configura regras e ações; (b) Ideia geral de AutoElastic

A. Arquitetura do Modelo

A Figura 2 ilustra a arquitetura de componentes de AutoElastic e o mapeamento de MVs. Primeiramente, o Gerente AutoElastic pode ser mapeado para uma MV dentro da nuvem ou atuar como um programa fora dela. Essa flexibilidade é atingida através do uso da API do *middleware* de nuvem usado. Uma vez que normalmente aplicações de alto desempenho são intensivas quanto ao uso de CPU, optou-se por criar um processo por MV e n MVs por nó, onde n é o número de núcleos de processamento (*cores*) que o nó possui. Essa abordagem está baseada nos trabalhos de Lee et al [4], onde se busca explorar uma melhor eficiência ($\frac{Speedup(p)}{p}$, para p núcleos) para a execução da aplicação. Num primeiro momento, AutoElastic suporta aplicações desenvolvidas segundo o modelo 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 [2].

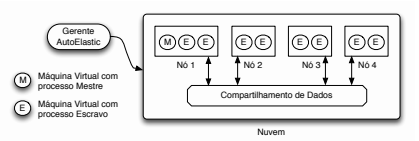

Figura 2. Arquitetura onde os Nós possuem dois núcleos de processamento

O Gerente AutoElastic monitora os Nós e toma as decisões de elasticidade. O usuário pode informar um arquivo de SLA com o mínimo e o máximo de Nós para a execução de sua aplicação. Caso esse arquivo não seja informado, assume-se que o número máximo permitido de Nós é o dobro daquele informado no lançamento. As operações de elasticidade realizadas devem chegar ao conhecimento da aplicação de alguma maneira. Para tal, foi modelada uma comunicação entre as MVs e o Gerente AutoElastic através de uma área de dados compartilhada que pode ser viabilizada, por exemplo, via compartilhamento NFS (Network File System), *middleware* orientado a mensagens, como JMS (Java Message Service) ou AMQP (Advanced Message Queuing Protocol), ou espaço de tuplas (como JavaSpaces).

Assim como nos trabalhos de Imai et al [5] e Chiu et al [6], o monitoramento do gerente para as ações de elasticidade é dado de forma periódica. Assim, de tempos em tempos analiza-se a métrica, que é comparada a um valor mínimo e outro máximo de *threshold*. A Figura 3 apresenta o modelo reativo de elasticidade de AutoElastic. Quanto às condições, elas trabalham com uma métrica chamada PC (Predição de Carga), que é baseada numa série que leva em conta a carga C em cada um dos pontos de monitoramento numa ordem da mais nova para a mais antiga. A carga C é dada pela média aritmética da carga de CPU de todas as MVs em execução. PC é encontrada pela fórmula de recorrência abaixo.

$$PC(i) = \begin{cases} C(i) & se\ i = 0 \\ \frac{1}{2}PC(i-1) + \frac{1}{2}C(i) & se\ i > 0 \end{cases}$$

REGRA1: if CONDIÇÃO1 então AÇÃO1
REGRA2: if CONDIÇÃO2 então AÇÃO2

CONDIÇÃO1: PC(i) > threshold1, onde i é a última observação do sistema de monitoramento.
CONDIÇÃO2: PC(i) < threshold2, onde i é a última observação do sistema de monitoramento.

AÇÃO1: Aloca um novo nó e lança n VMs, onde n é o número de núcleos de processamento no nó.
AÇÃO2: Finaliza as instâncias que estão rodando em um nó e depois o consolida.

Figura 3. Modelo de elasticidade reativa de AutoElastic

B. Modelo de Aplicação Paralela

A Figura 4 (a) apresenta uma aplicação mestre-escravo suportada por AutoElastic. Ela se caracteriza pelo caráter iterativo, onde o mestre possui uma série de tarefas, as captura sequencialmente e paraleliza uma a uma para processamento nos processos escravos. Essa captura de trabalhos

é evidenciada no *loop* externo da Figura 4 (a). AutoElastic trabalha com as seguintes diretivas baseadas na Interface de MPI 2.0: (i) publicar e despublicar 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 onde 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. O lançamento de uma máquina virtual acarreta automaticamente na execução de um processo escravo, que requisita uma conexão com o mestre.

```
1.  Para (j=0; j< total_trabalho; j++)
2.  {
3.      tamanho = publica_portas(portas);
4.      Para (i=0; i< tamanho; i++)
5.      {
6.          aceita_conexao(escravos[i], portas[i]);
7.      }
8.      calcula_carga(tamanho, trabalho[j], intervalos);
9.      Para (i=0; i< tamanho; i++)
10.     {
            tarefa = monta_tarefa(trabalho[j], intervalos[i])
11.         envia_assincrono(escravos[i], tarefa);
12.     }
13.     Para (i=0; i< tamanho; i++)
14.     {
15.         recebe(escravos[i], resultados[i];
16.     }
17.     grava_resultado(trabalho[j], resultados);
18.     Para (i=0; i< tamanho; i++)
19.     {
20.         desconexao(escravos[i]);
21.     }
22.     despublica_portas(portas);
23. }
```

(a)

```
1.  mestre = procura(endereco_mestre, servico_nomes);
2.  porta = monta_porta(endereco_IP, id_VM);
3.  Para (sempre)
4.  {
5.      requisita_conexao(mestre, porta);
6.      recebe(mestre, tarefa);
7.      resultado = computa(tarefa);
8.      envia(mestre, resultado);
9.      desconexao(mestre);
10. }
```

(b)

Figura 4. Modelo de aplicação suportado por AutoElastic: (a) pseudocódigo do mestre e; (b) pseudocódigo do processo escravo

O método da linha 2 do código do mestre verifica um arquivo de configuração ou argumentos passados para o programa que informam identificadores de máquinas virtuais e endereços IP de cada um dos processos. Com base nisso, 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, onde o envio de dados para os escravos é de forma não bloqueante e a recepção é bloqueante. O fato de assumir programas com um *loop* 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. A transformação da aplicação mostrada na Figura 4 em outra elástica pode ser feita em nível PaaS através de uma das seguintes maneiras: (i) numa implementação orientada a objetos, incluir um método cuja funcionalidade seja nula entre as linhas 2 e 3 do código do mestre, o qual é sobrescrito para gerir a elasticidade; (ii) fazer um tradutor fonte-para-fonte que insira um código entre as linhas 2 e 3; (iii) desenvolvimento de um *wrapper* em linguagens procedurais para o método da linha 3.

A região de código de monitoramento verifica no diretório compartilhado se há alguma ação nova de AutoElastic. Se tivermos ocorrência de Ação1, 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. Embora o foco inicial de AutoElastic sejam aplicações mestre-escravo, a modelagem iterativa e o uso de diretivas de MPI 2.0 (no estilo de Sockets) facilitam a inclusão de processos e o reestabelecimento 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 usa um protocolo de três vias que sabidamente pode acarretar sobrecarga em aplicações de alto desempenho.

III. Implementação de Protótipo

Um protótipo foi implementado em Java usando o sistema OpenNebula. Foram criadas imagens de dois modelos de máquinas virtuais: uma para o processo mestre e outra para processos escravos. O Gerente AutoElastic utiliza a própria API Java de OpenNebula para as atividades de monitoramento e emprega a elasticidade nas modalidades 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[1] e informa a quantidade mínima e máxima de Nós para teste da aplicação. A seguir são descritos em detalhes algumas decisões técnicas de AutoElastic.

- Compartilhamento de dados - Implementado com NFS. Tecnicamente, AutoElastic usa SSH para se conectar a máquina gerenciadora da nuvem e a partir dali tem acesso ao diretório compartilhado NFS.
- Noção de carga - Em nível de protótipo, a carga C para a amostra i de monitoramento, denominada $C(i)$ na fórmula de recorrência apresentada, é dada pela média aritmética da carga de todas as CPUs em execução num determinado momento.
- Monitoramento periódico - Utilizou-se o valor de 30 segundos para o intervalo de medições de desempenho.
- Thresholds - Com base em trabalhos relacionados, optou-se pela adoção de 80% para o valor máximo e 40% para o mínimo.

IV. Modelagem da Aplicação Paralela e Metodologia de Avaliação

A aplicação usada nos testes realiza o cálculo de soma de dois vetores de mesmo tamanho. A aplicação utiliza sockets TCP para a comunicação entre o processo mestre e os processos escravos. O processo mestre realiza a leitura de x arquivos, cada um contendo dois vetores que serão somados. Considerando o tamanho do subvetor definido por n e o tamanho do vetor definido por t, cada arquivo processado contém q tarefas para serem distribuídas para os processos escravos, sendo que q é definido por $q = \frac{t}{n}$. O processo mestre envia para cada processo escravo uma tarefa por iteração e aguarda o retorno dos processos para o envio das próximas tarefas. Ao concluir todas as tarefas geradas pela leitura de um arquivo, o arquivo seguinte é processado. Considerando i a quantidade de iterações para o processamento de cada arquivo e p a quantidade de processos disponíveis, i é dado por:

$$i = \begin{cases} \frac{q}{p} & se\ mod(\frac{q}{p}) = 0 \\ \frac{q}{p} + 1 & se\ mod(\frac{q}{p}) > 0 \end{cases}$$

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

Em um cenário onde $\frac{q}{p}$ não retornar um número inteiro, na última iteração do processamento de um arquivo existirão um ou mais processos ociosos.

V. Avaliação e Análise de Resultados

Foram realizados testes com um arquivo SLA definindo a quantidade mínima e máxima de Nós em 2 e 4 respectivamente. Os seguintes cenários foram avaliados: (i) Execução em nuvem com quantidade fixa de recursos; (ii) Execução em nuvem utilizando AutoElastic com o SLA informado. Foram utilizados arquivos contendo três tamanhos diferentes de vetores: 500, 1000 e 10000. Em todas as execuções o tamanho dos subvetores foi definido em 500, variando a quantidade de tarefas que cada arquivo de tamanho diferente possui. A aplicação foi executada no ambiente formado por dois Nós e cinco MVs, das quais quatro executando processos escravo e uma executando o processo mestre.

Para o processamento dos arquivos no primeiro cenário, o tempo de execução médio foi de 22 minutos e 34 segundos, com um desvio padrão de 30 segundos. Foram observados três níveis de carga gerados na nuvem dependendo dos arquivos que estavam sendo processados. Isso se deve a quantidade de tarefas que cada arquivo gerou. Neste cenário, nenhuma operação de elasticidade foi executada. No segundo cenário, o tempo de execução médio observado foi de 19 minutos e 23 segundos, com um desvio padrão de 31 segundos. Assim como no primeiro cenário, os arquivos processados geraram três níveis de carga diferentes. Como neste cenário o AutoElastic monitorou estes níveis, foram realizadas operações de elasticidade nos momentos em que cargas ficaram fora da faixa definida pelos *thresholds*, respeitando o SLA. Uma breve comparação demonstra que a utilização do AutoElastic representou um ganho de 14,1% no desempenho da aplicação. Todas as operações de elasticidade ocorreram sem nenhuma ação do usuário.

VI. Trabalhos Relacionados

ElasticMPI propõe a elasticidade em aplicações MPI através da parada e relançamento delas no momento da reconfiguração de recursos [2]. A abordagem de ElasticMPI faz uma alteração no código fonte da aplicação de modo a inserir diretivas de monitoramento. Em adição, a abordagem de ElasticMPI faz uma alteração no código fonte da aplicação de modo a inserir diretivas de monitoramento. A elasticidade é mais explorada em nível de plataforma (IaaS) e de forma reativa. Nesse sentido, os trabalhos não são uníssonos quanto ao emprego de um *threshold* único para os testes. Por exemplo, é possível notar os seguintes valores: (i) 70% [7]; (ii) 75% [5]; (iii) 80% [8], [9]; (iv) 90% [10], [11], [12]. Esses valores tratam de limites superiores que quando ultrapassados, acionam ações de elasticidade de forma horizontal e/ou vertical.

VII. Conclusão

O presente trabalho descreveu o modelo AutoElastic e um protótipo que trabalha sobre OpenNebula. Apesar de simples, o protótipo possibilitou verificar que os resultados da elasticidade automática são promissores. Foi adotada uma abordagem periódica para o monitoramento, que verifica o estado global dos processos e Nós e aplica regras de elasticidade. Tais regras estão baseadas em *thresholds* que quando atingidos, disparam ações para redimensionamento da arquitetura. Trabalhos futuros incluem a utilização de uma arquitetura dinâmica de recursos e aplicações irregulares quanto à carga de trabalho em cada processo.

Referências

[1] N. Cook, D. Milojicic, and V. Talwar, "Cloud management," *Journal of Internet Services and Applications*, vol. 3, pp. 67–75, 2012.

[2] 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.

[3] A. Tanenbaum, *Computer Networks*, 4th ed. Upper Saddle River, New Jersey: Prentice Hall PTR, 2003.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] R. Bryant, A. Tumanov, O. Irzak, A. Scannell, K. Joshi, M. Hiltunen, A. Lagar-Cavilla, and E. de Lara, "Kaleidoscope: cloud micro-elasticity via vm state coloring," in *Proceedings of the sixth conference on Computer systems*, ser. EuroSys '11. New York, NY, USA: ACM, 2011, pp. 273–286. [Online]. Available: http://doi.acm.org/10.1145/1966445.1966471

[9] 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.

[10] 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

[11] W. Dawoud, I. Takouna, and C. Meinel, "Elastic virtual machine for fine-grained cloud resource provisioning," in *Global Trends in Computing and Communication Systems*, ser. Communications in Computer and Information Science, P. Krishna, M. Babu, and E. Ariwa, Eds. Springer Berlin Heidelberg, 2012, vol. 269, pp. 11–25.

[12] 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.