Explorando a Elasticidade em Nível de Programação

Guilherme Galante[1], **Luis Carlos Erpen De Bona**[2]

[1]Centro de Ciências Exatas e Tecnológicas
Universidade Estadual do Oeste do Paraná (UNIOESTE)
Caixa Postal 711 – 85.819-110 – Cascavel-PR

[2]Departamento de Informática
Universidade Federal do Paraná (UFPR)
Caixa Postal 19.081 – 81.531-980 – Curitiba-PR

guilherme.galante@unioeste.br, bona@inf.ufpr.br

Abstract. *Several mechanisms to provide elasticity are offered by public cloud providers and in academic works, however, we argue that these solutions present limitations in providing elasticity for specific applications models. In this paper we propose an approach for exploring elasticity, in which the control is embedded within application code and the actions are performed by application itself, based in its runtime requirements or internal events. The experimental evaluation shows that programming level elasticity enables adding new functionalities to applications, without the restrictions presented by traditional mechanisms.*

Resumo. *Diversos mecanismos para a exploração da elasticidade em nuvens computacionais tem sido propostos, no entanto, estes ainda apresentam uma série de limitações ao fornecer suporte a algumas classes de aplicações. Neste trabalho, propõe-se uma abordagem para o desenvolvimento de aplicações elásticas, na qual o controle da elasticidade é feito em nível de programação, permitindo que as ações de elasticidade sejam realizadas pela própria aplicação. A avaliação experimental comprova que o controle de elasticidade em nível de programação permite que novas funcionalidades sejam acrescidas a aplicações de diversos modelos, sem apresentar as restrições dos mecanismos de elasticidade atuais.*

1. Introdução

Nos últimos anos, a computação em nuvem tem atraído a atenção da indústria e do mundo acadêmico, tornando-se cada vez mais comum encontrar na literatura casos de adoção da nuvem por parte das empresas e instituições de pesquisa. Um dos principais motivos é a possibilidade de aquisição de recursos de uma forma dinâmica e elástica. De fato, a elasticidade é um grande diferencial e é atualmente vista como indispensável para o modelo de computação em nuvem [Owens 2010].

A elasticidade pode ser definida como a capacidade de um sistema de adicionar ou remover dinamicamente recursos computacionais utilizados por uma determinada aplicação ou usuário de acordo com a demanda [Herbst et al. 2013]. Os recursos podem incluir desde CPUs virtuais, memória, e armazenamento, até máquinas virtuais (MVs) completas. O conceito de elasticidade também pode ser estendido para aplicações. Uma aplicação elástica é aquela capaz de se adaptar às alterações na quantidade de recursos

ou de solicitar/liberar recursos de acordo com a sua necessidade. Para que se possa tirar proveito da elasticidade, é necessário que tanto a arquitetura quanto a aplicação sejam capazes de suportá-la de alguma forma.

De acordo com o estado-da-arte do assunto [Galante e Bona 2012, Righi 2013], a elasticidade oferecida pelos mecanismos atuais baseia-se na variação do número de máquinas virtuais empregadas pelos componentes da aplicação e no uso de balanceadores de carga para dividir a carga de trabalho entre os diversos servidores virtualizados. O controle da elasticidade é realizado por meio de um conjunto de regras e condições para decidir sobre a adição ou remoção de recursos. Essa regras baseiam-se em informações provenientes de um sistema de monitoramento que coleta dados sobre a carga de trabalho gerada externamente (número de clientes, conexões, etc.) e sobre a utilização de recursos das máquinas virtuais [Vaquero et al. 2011]. Exemplos destes mecanismos são os fornecidos pelos provedores de nuvem pública Amazon WS, GoGrid e Rackspace.

Esses mecanismos têm sido usados com sucesso em aplicações cliente-servidor (servidores web, e-mail e banco de dados, por exemplo) com o intuito de evitar as questões relacionadas ao provisionamento excessivo ou insuficiente de recursos [Chieu et al. 2009], no entanto, ainda apresentam uma série de limitações para fornecer elasticidade para aplicações que não são construídas de acordo com o modelo suportado.

Para estas aplicações, a simples adição de máquinas virtuais e o uso de balanceadores de carga são inefetivos, uma vez que tais aplicações não são aptas a detectar e utilizar recursos adicionais, já que não foram construídas com estas características. Em muitas aplicações as cargas de trabalho são definidas por arquivos de entrada e parâmetros de configuração em vez de requisições externas. Isso torna difícil estimar se recursos devem ou não ser alocados, considerando que não há informações de cargas externas. Além disso, deve-se considerar que os sistemas de monitoramento não conseguem coletar informações sobre os eventos gerados internamente pela aplicação, e dessa forma, não se pode estimar de modo acurado se novos recursos são realmente necessários.

Considerando o exposto, este trabalho tem como objetivo propor uma abordagem alternativa para a exploração de elasticidade para aplicações que não se adequam aos mecanismos de elasticidade atuais. Nesta abordagem, o controle da elasticidade é definido em *nível de programação*, o que significa que toda lógica de controle é incorporada ao código-fonte da aplicação. Com isso, torna-se possível desenvolver aplicações elásticas capazes de gerenciar de modo adequado a alocação de seus próprios recursos, uma vez que o controle da elasticidade faz parte de sua lógica e que os elementos internos da aplicação passam a ser considerados.

A exploração da elasticidade em nível de programação é validada por um conjunto de experimentos, nos quais a abordagem proposta é utilizada com sucesso na alocação dinâmica de processadores virtuais (VCPUs) para uma aplicação de transferência de calor, na adição de nodos em uma aplicação de montagem de genomas mestre-escravo, e na alocação dinâmica de memória em uma aplicação sintética.

O restante do trabalho é organizado da seguinte forma. A Seção 2 apresenta uma visão geral da exploração da elasticidade em nuvens computacionais. A Seção 3 apresenta a abordagem proposta neste trabalho, na qual a exploração da elasticidade ocorre em nível

de programação. Na Seção 4 realiza-se a avaliação experimental. Por fim, a Seção 5 conclui este trabalho.

2. Elasticidade em Nuvens: Visão Geral

A elasticidade é uma das principais características da computação em nuvem. Esta característica consiste na capacidade de adicionar ou remover recursos, sem interrupções e em tempo de execução, de acordo com uma demanda específica. A capacidade de elasticamente expandir e contrair a base de recursos é interessante tanto para o provedor quanto para o usuário final da nuvem.

Do ponto de vista do provedor, a elasticidade garante um melhor uso dos recursos de computação, fornecendo economia de escala e permitindo que mais usuários possam ser atendidos simultaneamente, uma vez que os recursos liberados por um usuário podem instantaneamente ser alocados por outro. Da perspectiva do usuário, a elasticidade tem sido utilizada para diversos fins, tais como manutenção da qualidade de serviço [Roy et al. 2011], complementação de recursos locais [Marshall et al. 2010, Calheiros et al. 2011], redução de custos [Sharma et al. 2011], economia de energia elétrica [Shen et al. 2011], entre outros.

Dependendo da forma como a nuvem implementa o provisionamento de recursos, pode-se classificar a sua elasticidade em horizontal ou vertical [Vaquero et al. 2011]. Uma nuvem com elasticidade horizontal possibilita apenas a adição ou a remoção dinâmica de MVs da plataforma de computação alocada pelo usuário. Por sua vez, a elasticidade vertical é caracterizada pela possibilidade de se alterar a capacidade de MVs em execução. Tipicamente, a elasticidade vertical é implementada através da adição ou remoção de CPUs e memória, mas também pode ser utilizada no contexto de redes e armazenamento.

Em trabalho prévio [Galante e Bona 2012], diversos mecanismos de elasticidade foram descritos e classificados. Grande parte dos mecanismos analisados tem como objetivo escalar aplicações cliente-servidor (servidores web, e-mail, banco de dados, *streaming*) com o intuito de evitar as questões relacionadas com o provisionamento excessivo ou insuficiente de recursos. Em geral, estes mecanismos exploram apenas elasticidade horizontal em conjunto com técnicas de balanceamento de carga.

Em contrapartida, alguns trabalhos acadêmicos têm desenvolvido soluções que permitem o desenvolvimento de aplicações elásticas para outros modelos de aplicações. É possível encontrar trabalhos com foco em *workflows* [Byun et al. 2011], MapReduce [Chohan et al. 2010, Iordache et al. 2012], MPI (*Message Passing Interface*) [Raveendran et al. 2011] e aplicações mestre-escravo [Rajan et al. 2011].

Em síntese, ao se analisar o estado-da-arte, é possível notar que as soluções ainda são voltadas a algum modelo de programação específico e não permitem uma ampla exploração da elasticidade oferecida pelo modelo de computação em nuvem. Para que isso seja possível, é necessário fornecer uma solução de elasticidade que (1) permita que as características de cada aplicação sejam levadas em conta, que (2) ofereça suporte para que as aplicações possam solicitar, detectar e usar recursos elásticos, e que (3) forneça mecanismos para a exploração da elasticidade tanto vertical, quanto horizontal. Neste sentido, propõe-se na próxima seção uma abordagem para a exploração da elasticidade de modo amplo e independente do modelo de aplicação.

3. Elasticidade em Nível de Programação

Nesta seção, propõe-se uma abordagem para a exploração da elasticidade, na qual o controle da elasticidade é feito em nível de programação, ou seja, o controlador de elasticidade é incorporado ao código-fonte, permitindo que as ações de alocação e desalocação de recursos partam da própria aplicação, sem a necessidade de mecanismos externos ou de interação com o usuário.

Conforme ilustrado na Figura 1, ao se mover o controlador da elasticidade para dentro da aplicação, este passa a ter acesso a todas as informações internas, enquanto que os mecanismos baseados em monitoramento coletam apenas informações sobre as cargas de trabalho e sobre o estado da máquina virtual.

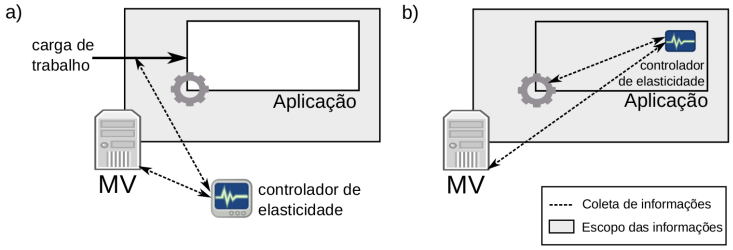

Figura 1. a) Controle externo. b) Controle incorporado na aplicação.

A possibilidade de considerar a alocação de recursos como parte da lógica do programa dá origem a um novo paradigma para o projeto e o desenvolvimento de aplicações. Neste paradigma, os recursos passam a ser tratados como elementos variáveis de um programa, podendo ser instanciados e modificados ao longo da execução. Isso permite que o programador desenvolva e integre à aplicação controladores de elasticidade que consideram as características particulares das aplicações, tais como modelo de programação, eventos internos, dados de entrada e seus parâmetros de configuração.

Como consequência, funcionalidades inéditas, não previstas nos mecanismos de elasticidade atuais, podem ser agregadas às aplicações. Pode-se desenvolver aplicações dinâmicas e flexíveis que adaptam o seu próprio ambiente de execução de acordo com sua estrutura lógica e demandas, de modo a aprimorar seu desempenho, melhorar o uso dos recursos, reduzir o custo de execução, ou ainda, tirar vantagem de recursos ociosos ou de baixo custo.

Considerando que, na abordagem proposta, a coleta de informações e as ações de elasticidade fazem parte da implementação da aplicação, devem ser oferecidos mecanismos apropriados para tais tarefas. Neste trabalho, propõe-se o uso de *primitivas de elasticidade*, que correspondem a um conjunto de funções básicas que permitem a comunicação com a infraestrutura de nuvem subjacente para a solicitação ou liberação de recursos, bem como coletar informações do ambiente virtual.

O conjunto de primitivas deve ser abrangente o suficiente para permitir a ampla exploração da elasticidade. Isso significa que devem ser oferecidas primitivas para o emprego de elasticidade horizontal, permitindo a alocação e desalocação de máquinas virtuais completas, bem como para o uso de elasticidade vertical, permitindo a reconfiguração das máquinas pela adição ou remoção dos componentes que as compõem, tais como

VCPUs, memória e disco. Deve-se contemplar também primitivas para a coleta de informações do ambiente virtual e da infraestrutura da nuvem. Tais informações são indispensáveis para a elaboração dos controladores de elasticidade, uma vez que é por meio destas que determina-se a necessidade de novos recursos e se há disponibilidade para alocação.

Revisitando as limitações apresentadas no final da Seção 2, pode-se afirmar que todas elas são superadas na abordagem proposta. Ao mover o controlador de elasticidade para dentro do código-fonte da aplicação pode-se construir controladores que levam em consideração as demandas e os eventos gerados internamente nas aplicações, eliminando a limitação (1). A limitação (2) é superada uma vez que as aplicações não são só capazes de usar recursos adicionais, mas também ganham a capacidade de solicitar ou liberar seus próprios recursos. Por fim, a limitação (3) é abolida ao se considerar que as primitivas devem oferecer a possibilidade de explorar a elasticidade tanto horizontal quanto vertical.

3.1. Cloudine

Para oferecer suporte para a construção e a execução de aplicações elásticas utilizando a abordagem proposta, desenvolveu-se o *framework* Cloudine. O *framework* compreende dois componentes principais: o Ambiente de Execução (*runtime environment*) e a API de Elasticidade. O Ambiente de Execução é o componente que gerencia o provisionamento dinâmico de recursos e realiza toda a interação entre as aplicações elásticas (via API) e infraestrutura de nuvem. É por meio deste componente que as solicitações são processadas e repassadas à nuvem subjacente.

Por sua vez, a API de Elasticidade fornece um conjunto de primitivas que permitem a construção de aplicações elásticas. As primitivas implementam a comunicação entre a aplicação e o Ambiente de Execução, que é utilizado para a realização da alocação ou liberação de recursos. Até o presente momento, a API implementada suporta a linguagem C/C++ e oferece 12 primitivas para a alocação elástica de VCPUs, memória, e máquinas virtuais completas, bem como para coleta de informações sobre o ambiente virtual e sobre a infraestrutura da nuvem. As primitivas são apresentadas na Tabela 1.

Tabela 1. Primitivas implementadas pela API de Elasticidade

Função	Descrição
`int clne_add_vcpu(int N)`	Adiciona N VCPUs para a MV na qual a aplicação está sendo executada. Retorna 1 em caso de sucesso, 0 caso contrário.
`int clne_rem_vcpu(int N)`	Remove N VCPUs da MV na qual a aplicação está sendo executada. Retorna 1 em caso de sucesso, 0 caso contrário.
`int clne_add_node(int N)`	Adiciona N nodos ao ambiente virtual (cluster). Retorna 1 em caso de sucesso, 0 caso contrário. Esta função também cria (ou atualiza) um arquivo contendo os nomes e endereços de IP dos nodos que compõem o cluster.
`int clne_rem_node(int N)`	Remove o nodo atual do ambiente virtual (cluster). Retorna 1 em caso de sucesso, 0 caso contrário.
`int clne_add_memory(long N)`	Adiciona N Megabytes de memória para a MV na qual a aplicação está sendo executada. Retorna 1 em caso de sucesso, 0 caso contrário.
`int clne_rem_memory(long N)`	Remove N Megabytes de memória da MV na qual a aplicação está sendo executada. Retorna 1 em caso de sucesso, 0 caso contrário.
`int clne_get_freemem()`	Retorna a quantidade de memória livre da máquina física que hospeda a MV na qual a aplicação está sendo executada.
`int clne_get_maxmem()`	Retorna a quantidade total de memória da máquina física que hospeda a MV.
`int clne_get_mem()`	Retorna a quantidade de memória livre da MV na qual a aplicação está sendo executada.
`int clne_get_freecpu()`	Retorna a quantidade de CPUs livres na máquina física que hospeda a MV.
`int clne_get_maxcpu()`	Retorna a quantidade total de CPUs da máquina física que hospeda a MV.
`int clne_get_vcpus()`	Retorna a quantidade de CPUs da MV na qual a aplicação está sendo executada.

No Cloudine, uma aplicação elástica é construída a partir da inserção de primitivas de elasticidade, que quando executadas comunicam-se com Ambiente de Execução para enviar uma requisição. Esta requisição é então convertida pelo Ambiente de Execução em um conjunto de comandos para a nuvem. A Figura 2 ilustra o uso das primitivas na alocação dinâmica de recursos. Neste exemplo, solicita-se a adição de 2 VCPUs, que em seguida são alocadas à máquina virtual em que a aplicação está sendo executada.

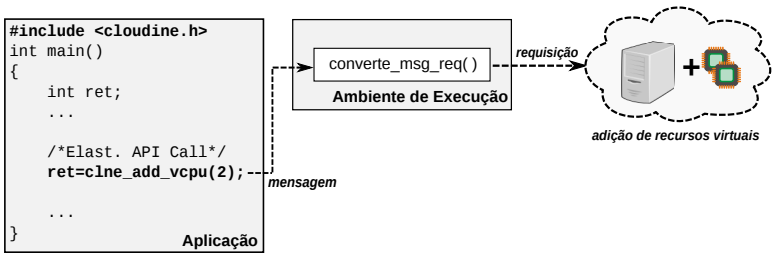

Figura 2. Exemplo de funcionamento das primitivas de elasticidade.

4. Avaliação Experimental

Nesta seção apresenta-se um conjunto de experimentos nos quais emprega-se o controle de elasticidade em nível de programação para o desenvolvimento de aplicações elásticas.

Inicialmente, na Seção 4.1, descreve-se o ambiente computacional utilizado. Nas seções subsequentes, apresentam-se um alguns experimentos para a validação do *framework*, onde utiliza-se duas aplicações distintas e demonstrando a viabilidade da abordagem proposta.

4.1. Ambiente Computacional

Os experimentos foram executados em uma nuvem privada OpenNebula versão 3.4 com suporte a virtualização Xen. A plataforma OpenNebula originalmente não suporta elasticidade vertical, e dessa forma foi necessário alterá-la para incluir essa funcionalidade, incluindo novas funções à interface e modificações no banco de dados. O uso de OpenNebula e Xen permite utilizar todas as funcionalidades oferecidas na implementação do Cloudine, tornando possível a exploração de elasticidade vertical e horizontal.

O hardware utilizado é composto por um servidor equipado com três processadores AMD Opteron 6136, com 8 cores de 2.40 GHz, 98 GB RAM e 2 TB de armazenamento. Foram utilizados 4 cores e 10 GB RAM para o domínio Xen dom0 e o restante dos núcleos e da memória foram disponibilizados para o uso nas máquinas virtuais. O sistema operacional utilizado em todas as máquinas, física e virtuais, é o Ubuntu Server 12.04 com kernel 3.2.0-29.

4.2. Transferência de calor *multithread*

Um problema clássico de aplicação de métodos numéricos em fenômenos físicos é a transferência de calor em uma placa plana homogênea. Considerando que todos os pontos internos da placa estejam a uma temperatura inicial diferente das temperaturas das bordas, o problema consiste em determinar a temperatura em qualquer ponto interno da

placa em um dado instante de tempo [Lienhard e Lienhard 2008]. A solução implementada consiste na montagem de um sistema de equações e sua resolução (via Gradiente Conjugado) para cada passo de tempo de simulação. Ambas as etapas são paralelizadas usando OpenMP.

Com esta aplicação, realizou-se um experimento no qual empregam-se as primitivas de elasticidade para permitir que recursos ociosos na nuvem (CPUs) sejam utilizados para a redução do tempo de execução. Foi necessário modificar o código OpenMP original com primitivas de elasticidade, conforme apresentado na Figura 3. A aplicação possui um laço iterativo que determina a evolução temporal da simulação. No início deste laço, uma primitiva foi inserida no código-fonte para verificar se existem CPUs ociosas na máquina física. Caso CPUs estejam disponíveis, elas são alocadas para a MV. O número de *threads* OpenMP ativas foi definido como o mesmo número de VCPUs da MV. Além disso, foi desenvolvido uma aplicação auxiliar, cujo o objetivo é simplesmente liberar V VCPUs alocadas para a MV onde é executada a cada T segundos.

```
transferência_calor()
    ...
    para cada iteração:
        CPUs_livres=clne_get_freecpu()

        se CPUs_livres>0
            clne_add_vcpu(CPUs_livres)
            threads=threads+CPUs_livres

        calcula_calor_OpenMP(threads)
```

Figura 3. Versão elástica da aplicação de transferência de calor OpenMP.

A Figura 4 apresenta os resultados da alocação elástica de recursos. Foram executadas 100 iterações da aplicação de transferência de calor em um domínio quadrado com 8.192×8.192 células. No início do teste, foram atribuídas duas VCPUs para a MV da aplicação OpenMP e 18 VCPUs para a aplicação auxiliar. Na Figura 4, é possível observar o resultado de duas execuções usando recursos elásticos. Na primeira, uma VCPU é liberada pela aplicação auxiliar a cada 120 segundos que é alocada pela aplicação OpenMP, que termina a sua execução com 11 VCPUs. A aplicação apresenta um tempo de execução de aproximadamente 1.205 segundos, sendo 5,3 mais rápida do que a execução utilizando apenas uma VCPU (3.220 segundos).

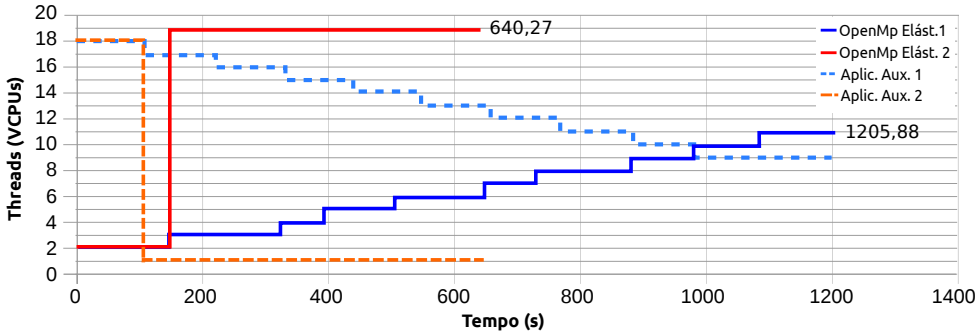

Figura 4. Alocação elástica de recursos ociosos.

Na segunda execução, 17 VCPUs são liberadas pela aplicação auxiliar após 120 segundos. Algum tempo depois, todos esses recursos são alocados pela aplicação elástica, que passa a executar usando 19 VCPUs. Neste caso, o tempo de execução foi de 640,27 segundos, apresentando *speedup* de 10 vezes. Em ambos os casos, a sobrecarga gerada pelas primitivas de elasticidade foi de aproximadamente 4% do tempo total de execução.

Note que a versão não-elástica da aplicação OpenMP no mesmo cenário (com apenas 2 VCPUs disponíveis em sua inicialização) apresenta tempo médio de execução de 3.220 segundos, sendo 2,67 e 5,02 vezes mais lento do que a aplicação elástica nos dois casos apresentados. É importante observar que a versão não-elástica usando 19 *threads* apresenta tempo de execução de 429 segundos. No entanto, se considerarmos o tempo de espera para adquirir todos as 19 VCPUs necessárias para executar a aplicação, o tempo total (tempo de espera somado ao tempo de execução) seria de 2.469 segundos, considerando a liberação de uma VCPU cada 120 segundos, e 669 segundos se todos os recursos se tornassem disponíveis após 120 segundos.

Analisando ambos os cenários, é possível ver as vantagens de se empregar a elasticidade para explorar o uso de recursos ociosos. A exploração da elasticidade só foi possível por que a aplicação foi instrumentada adequadamente para solicitar e utilizar recursos adicionais. Solução similar não pode ser alcançada com os mecanismos atuais de elasticidade.

4.3. *Scalable Assembler at Notre Dame* - SAND

Para demonstrar o uso da elasticidade horizontal, o Cloudine foi empregado em conjunto com o *framework* Work Queue [Rajan et al. 2011] na adaptação de uma aplicação de montagem de genomas - SAND.

O SAND é um conjunto de módulos para a montagem do genoma construídos para a execução em ambientes de computação distribuída, tais como *clusters*, grades e nuvens [Moretti et al. 2012]. A aplicação é composta por duas fases principais: seleção de candidatos e alinhamento de sequências. A etapa de seleção de candidatos sugere pares de leituras que podem se sobrepor. Toma-se como entrada um conjunto de sequências e gera-se um conjunto de pares de candidatos para a fase de alinhamento. A fase de alinhamento utiliza como entrada o conjunto de sequências e uma lista de pares de sequências candidatas criados na etapa anterior, e gera como saída um conjunto de registros de sobreposição indicando quais sequências alinham-se adequadamente. Este conjunto de pares é utilizado no estágio final do montador de genomas.

Ambas as etapas são implementadas como uma aplicação mestre-escravo usando o *framework* Work Queue. Neste *framework*, o processo mestre é responsável por enviar um módulo executável (binário) e os arquivos de entrada para os escravos. Os escravos invocam o módulo executável para processar os dados enviados, armazenando os resultados localmente na máquina onde são executados. Quando a tarefa é concluída, os resultados são enviados ao mestre, que verifica o resultado e o armazena permanentemente.

Em sua especificação original, o mestre deve ser instanciado manualmente em uma máquina e os trabalhadores podem ser executados em nós de um *cluster*, grade ou nuvem, de modo manual ou usando um sistema de lote, como o Condor[1].

[1] `http://www.cs.wisc.edu/condor`

Neste experimento, o processo mestre de ambas as etapas foi modificado para alocar de modo automático novos escravos usando recursos da nuvem OpenNebula, como apresentado no pseudo-algoritmo da Figura 5. O processo mestre verifica constantemente se existem recursos disponíveis para a criação de novas MVs para executar os escravos. Em caso positivo, uma nova MV é criada e o próprio mestre inicia o trabalhador na máquina. Quando a tarefa é concluída, todas as MVs utilizadas pelos processos escravos são finalizadas.

```
mestre_SAND()
    ...
    enquanto(1):
        se recursos_livres
            clne_add_node(1)
            //aguarda a inicializacao da MV

        upload_trabalhador_MV
        inicia_trabalhador_MV
    ...
```

Figura 5. Alocação de MVs e inicialização de trabalhadores.

A Figura 6 apresenta uma linha do tempo da alocação das máquinas virtuais utilizadas no processamento do genoma do mosquito *Anopheles gambiae Mopti*[2], na qual os círculos representam a criação das MVs, os quadrados representam a desalocação das MVs, e as estrelas representam o início/fim das fases de processamento.

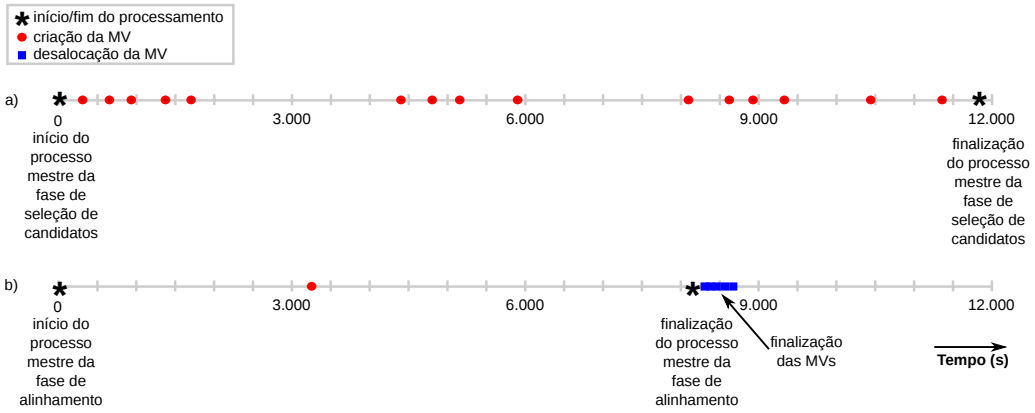

Figura 6. SAND: Alocação dinâmica de máquinas virtuais.

No início da execução (Figura 6 a)), o processo mestre da fase de seleção de candidatos é iniciado e 15 máquinas virtuais são criadas para hospedar processos escravos. A etapa de seleção é finalizada após 11.800 segundos. Na sequência, a fase de alinhamento é iniciada (Figura 6 b)). As 15 MVs anteriormente criadas são também utilizadas nesta fase e um nova máquina é instanciada após 3.300 segundos. Ao término do processamento, todas as MVs são finalizadas e os recursos são liberados. O *speedup* alcançado na fase de alinhamento foi de 12 vezes (se comparado à execução usando um processo escravo), de acordo com os dados apresentados na interface da aplicação.

[2]http://www3.nd.edu/~ccl/software/sand/

Como os resultados mostram, o uso do *framework* Cloudine mostrou-se eficaz na construção dos mecanismos para provisionamento automático de escravos para o SAND. O mecanismo implementado permite que o usuário inicie a aplicação utilizando o mínimo de recursos, e que esta aloque posteriormente seus próprios recursos sem a necessidade da interação do usuário ou o uso de escalonadores (como originalmente especificado).

Note que a política de alocação de recursos adicionais neste experimento foi baseada na disponibilidade de recursos, mas pode ser modificada de acordo com condições desejadas. Por exemplo, é possível alocar recursos seguindo uma função custo, ou ainda, de acordo com uma restrição de tempo de resposta (*deadline*).

4.4. Alocação dinâmica de memória

A capacidade de modificar dinamicamente a memória de uma MV em tempo de execução representa um recurso importante para aplicações com requisitos dinâmicos de memória. Este recurso permite manter memória suficiente para alocar os dados da aplicação na memória residente e, consequentemente, preservar o desempenho da aplicação. Além disso, a memória não utilizada pode ser liberada quando não for mais demandada pela aplicação.

Nesta seção, apresenta-se um experimento usando as primitivas de elasticidade e compara-se seus resultados com os resultados obtidos com o mecanismo de solução baseada em monitoramento proposto no trabalho de Moltó et al. (2013) . Para tal, utilizou-se uma aplicação sintética para gerar padrões distintos de uso de memória por meio de alocações e desalocações de memória.

Os testes foram realizados com dois cenários distintos: (1) carga de trabalho crescente e (2) de carga de trabalho oscilante. No primeiro, a carga de trabalho é aumentada a cada duas iterações, e no segundo, as demandas por memória alternam entre altas e baixas a cada duas iterações. Em ambos os testes, foram empregadas duas MVs com 2 VCPUs e, inicialmente, 2 GB RAM.

Primeiramente, avaliou-se o uso das rotinas de alocação dinâmica de memória fornecidas pelo Cloudine. A aplicação foi modificada através da substituição das rotinas *malloc*'s e *free*'s pelas rotinas de alocação elástica de memória. As Figuras 7 a) e b) mostram os resultados usando esta abordagem. Em ambos os cenários, as primitivas trabalharam de forma eficiente na alocação e desalocação de recursos, fornecendo memória suficiente para executar a aplicação. Também é possível observar que toda a aplicação foi mantida na memória residente durante todo o período de execução. A diferença entre a memória alocada (linha vermelha) e a memória utilizada pela aplicação (linha preta) é bem próximo dos 2 GB inicialmente atribuídos à VM. Com esta abordagem, o tempo total de execução foi de aproximadamente 267 segundos para o primeiro cenário e 278 segundos para o segundo.

No segundo teste, utilizou-se o sistema baseado no monitoramento das MVs proposto por Moltó et al. Os resultados obtidos com este mecanismo são apresentados na Figura 7 c) e d). No primeiro cenário, o mecanismo obteve êxito para proporcionar a memória necessária para a aplicação, mesmo na maioria das vezes não mantendo toda a aplicação na memória residente, o que afetou o desempenho da aplicação. O tempo total de execução neste cenário foi de aproximadamente 322 segundos. No segundo cenário, o

mecanismo falhou ao alocar memória para o pico de carga de trabalho e a aplicação foi abortada no início da segunda iteração.

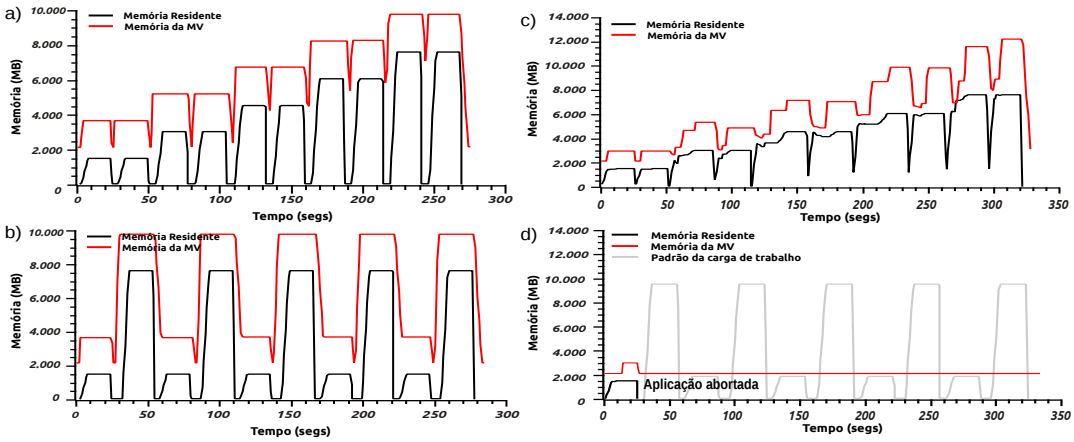

Figura 7. Alocação dinâmica de memória: comparação de abordagens.

5. Conclusão

Este trabalho teve como objetivo apresentar uma nova abordagem para o desenvolvimento de aplicações elásticas, sob a motivação de que que os mecanismos presentes no estado-da-arte apresentam ainda uma série de limitações.

Na proposta apresentada, a elasticidade é controlada em nível de programação, o que significa que todo o controle da elasticidade é incorporado ao código-fonte e passa a fazer parte da lógica da aplicação. Essa possibilidade de considerar a alocação de recursos como parte da lógica do programa dá origem a um novo paradigma para o projeto e o desenvolvimento de aplicações. Neste paradigma, os recursos passam a ser tratados como elementos variáveis de um programa, podendo ser instanciados e modificados ao longo da execução. Essa característica torna possível o desenvolvimento de aplicações elásticas capazes de adaptar o seu próprio ambiente de execução de acordo com suas demandas ou de modo a adaptá-lo a um cenário específico, relacionado a fatores como custo e desempenho.

De fato, os resultados apresentados comprovam que o controle de elasticidade em nível de programação é uma alternativa eficiente e eficaz para o desenvolvimento de aplicações elásticas. Foram desenvolvidas aplicações elásticas em diferentes modelos com funcionalidades que não poderiam ser agregadas caso mecanismos de elasticidade tradicionais tivessem sido empregados.

Referências

Byun, E., Kee, Y., Kim, J., e Maeng, S. (2011). Cost optimized provisioning of elastic resources for application workflows. *Future Generation Computer Systems*, 27(8):1011–1026.

Calheiros, R. N., Vecchiola, C., Karunamoorthy, D., e Buyya, R. (2011). The aneka platform and qos-driven resource provisioning for elastic applications on hybrid clouds. *Future Generation Computer Systems*, 28(6):861–870.

Chieu, T. C., Mohindra, A., Karve, A. A., e Segal, A. (2009). Dynamic Scaling of Web Applications in a Virtualized Cloud Computing Environment. In *Proceedings of the 2009 IEEE International Conference on e-Business Engineering*, ICEBE 2009, pages 281–286. IEEE.

Chohan, N., Castillo, C., Spreitzer, M., Steinder, M., Tantawi, A., e Krintz, C. (2010). See spot run: using spot instances for mapreduce workflows. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, HotCloud'10, pages 1–7. USENIX.

Galante, G. e Bona, L. C. E. (2012). A survey on cloud computing elasticity. In *Proceedings of the International Workshop on Clouds and eScience Applications Management*, CloudAM'12, pages 263–270. IEEE.

Herbst, N. R., Kounev, S., e Reussner, R. (2013). Elasticity in cloud computing: What it is, and what it is not. In *Proceedings of the 10th International Conference on Autonomic Computing*, ICAC'13, pages 23–27. USENIX.

Iordache, A., Morin, C., Parlavantzas, N., e Riteau, P. (2012). Resilin: Elastic MapReduce over Multiple Clouds. Rapport de recherche RR-8081, INRIA.

Lienhard, J. H. e Lienhard, J. H. (2008). *A Heat Transfer Textbook - 3rd ed.* Phlogiston Press: Cambridge, Massachusetts.

Marshall, P., Keahey, K., e Freeman, T. (2010). Elastic site: Using clouds to elastically extend site resources. In *Proceedings of the 10th International Conference on Cluster, Cloud and Grid Computing*, CCGRID'10, pages 43–52. IEEE.

Moltó, G., Caballer, M., Romero, E., e de Alfonso, C. (2013). Elastic memory management of virtualized infrastructures for applications with dynamic memory requirements. In *International Conference on Computational Science, ICCS'13*, volume 18 of *Procedia Computer Science*, pages 159–168.

Moretti, C., Thrasher, A., Yu, L., Olson, M., Emrich, S., e Thain, D. (2012). A framework for scalable genome assembly on clusters, clouds, and grids. *IEEE Transactions on Parallel and Distributed Systems*, 23(12):2189–2197.

Owens, D. (2010). Securing elasticity in the cloud. *Queue*, 8(5):10:10–10:16.

Rajan, D., Canino, A., Izaguirre, J. A., e Thain, D. (2011). Converting a high performance application to an elastic cloud application. In *Proceedings of the 3rd International Conference on Cloud Computing Technology and Science*, CLOUDCOM'11, pages 383–390. IEEE.

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

Righi, R. R. (2013). Elasticidade em cloud computing: conceito, estado da arte e novos desafios. *Revista Brasileira de Computação Aplicada*, 5(2):2–17.

Roy, N., Dubey, A., e Gokhale, A. (2011). Efficient autoscaling in the cloud using predictive models for workload forecasting. In *Proceedings of the 4th International Conference on Cloud Computing*, CLOUD'2011, pages 500–507. IEEE.

Sharma, U., Shenoy, P., Sahu, S., e Shaikh, A. (2011). A cost-aware elasticity provisioning system for the cloud. In *Proceedings of the 31st International Conference on Distributed Computing Systems*, ICDCS'11, pages 559–570. IEEE.

Shen, Z., Subbiah, S., Gu, X., e Wilkes, J. (2011). Cloudscale: elastic resource scaling for multi-tenant cloud systems. In *Proceedings of the 2nd Symposium on Cloud Computing*, SOCC'11, pages 5:1–5:14. ACM.

Vaquero, L. M., Rodero-Merino, L., e Buyya, R. (2011). Dynamically scaling applications in the cloud. *ACM Computer Communications Review*, 41:45–52.