Nebulous: A Framework for Scientific Applications Execution on Cloud Environments

Guilherme Galante, Luis Carlos E. de Bona
Departamento de Informática – Universidade Federal do Paraná (UFPR)
Caixa Postal 19.081 – 81.531-980 – Curitiba, PR
{ggalante, bona}@inf.ufpr.br

Abstract

This paper presents a framework, called Nebulous, designed to simplify the execution of MPI and OpenMP parallel applications in computational clouds. The framework aims to automate the process of deployment and execution, avoiding the direct user interaction with the cloud. The framework is built on a cloud middleware layer (e.g. OpenNebula, Eucalyptus or Nimbus) and consists of three components: the Resource Description Block, an Application Programming Interface and the Executor. According to the results, Nebulous allows users to deploy applications over varying numbers of nodes in a simple way, uniformly, scalably and with minimum effort.

1. Introdução

A computação em nuvem tem se mostrado, atualmente, uma plataforma viável para a execução de aplicações científicas. Alguns pesquisadores têm adotado este paradigma para a execução de seus experimentos *in silico*, movendo seus dados e aplicações de seus ambientes locais para a nuvem [24, 12, 23].

O uso da computação em nuvem pode ser atraente à comunidade científica por muitos aspectos, dentre os quais destacam-se o rápido fornecimento de recursos, a personalização dos ambientes computacionais e acesso privilegiado ao mesmo (*root*), a possibilidade de reproduzir experimentos, e a capacidade de obtenção de recursos de modo escalável e sob demanda.

É fato que as nuvens computacionais não substituem os supercomputadores ou *clusters* dedicados, mas podem ser vistas como uma alternativa para a execução de testes ou ainda fornecer recursos de computação para grupos de pesquisa, cujas demandas são temporárias, tornando inviável a adquisição de máquinas de maior porte [4, 22]. Além disso, a disponibilidade de *middleware* de nuvem, como o Nimbus, Eucalyptus e OpenNebula [19], e de software de virtualização como o Xen e o KVM [7], possibilita que as organizações construam nuvens privadas ou comunitárias para melhorar a utilização dos recursos de computação já disponíveis.

Apesar do constante desenvolvimento da área, executar aplicações científicas na nuvem ainda apresenta desafios [20]: (1) a implantação inicial das aplicações, (2) a execução propriamente dita, e (3) a transferência dos dados de e para a nuvem. Em geral, a implantação de um experimento na nuvem requer que o software seja instalado em uma imagem de máquina virtual. Essa imagem é, então, copiada para a nuvem, para que novas instâncias de máquinas virtuais possam ser criadas a partir desta e então a aplicação possa ser executada. O envio de dados de entrada para a aplicação, bem como a recepção dos resultados, também precisam ser tratadas explicitamente pelo usuário por meio de algum protocolo de rede. Esses desafios podem ser ainda mais relevantes quando se trata de aplicações paralelas ou distribuídas que dependem de um conjunto de máquinas virtuais que devem ser corretamente configuradas.

Considerando tais desafios, este trabalho apresenta uma solução para automatizar o processo de execução de aplicações em nuvens baseada no *framework* Nebulous. O *framework* foi projetado com dois objetivos: facilitar a implantação e a execução de aplicações científicas sequenciais e paralelas, desenvolvidas com OpenMP e MPI, em ambientes de nuvem; e fornecer escalabilidade em tempo de execução a estas.

O restante do trabalho está organizado da seguinte forma. A Seção 2 contextualiza e caracteriza as aplicações alvo do *framework*. A Seção 3 descreve com detalhes o Nebulous. A Seção 4 apresenta alguns testes e resultados obtidos com a utilização do *framework* proposto. Na Seção 4 alguns trabalhos relacionados são apresentados. Por fim, a Seção 5 conclui o trabalho e sugere algumas propostas para a continuação do trabalho.

2 Aplicações Científicas Paralelas

A Computação Científica é uma área de conhecimento que envolve a utilização de ferramentas computacionais para a solução de problemas científicos nas mais diversas áreas da ciência. Pesquisas climáticas e do sistema terrestre, dinâmica de fluidos, estudo de genomas e proteomas, química teórica, astrofísica, física de nanoestruturas e física de altas energias são alguns exemplos.

Na maioria dos casos, a solução destes problemas possui um elevado tempo de resposta, de modo que pode ser apropriado o uso de processamento paralelo para tornar a execução mais rápida. Além disso, a utilização de vários processadores na execução de uma aplicação viabiliza que novas e maiores instâncias do problema possam ser investigadas.

Com a popularização cada vez maior de hardware paralelo e tecnologias de software, os usuários se deparam com o desafio de escolher um paradigma de programação mais adequado para a arquitetura adjacente. Quando se trata de paralelização de aplicações científicas, duas ferramentas surgem como padrões *de-facto*: MPI, para memória distribuída, e OpenMP, para memória compartilhada.

O MPI (*Message Passing Interface*) [11] é uma especificação padrão para a troca de mensagens para ambientes paralelos. Existem diversas implementações do padrão MPI, tais como o MPICH e o OpenMPI, que disponibilizam um conjunto de rotinas de comunicação ponto-a-ponto e coletivas, bem como rotinas de gerenciamento de processos.

O OpenMP (*Open Multiprocessing*) [3] é uma interface de programação (API), portável, baseada no modelo de programação paralela arquiteturas de múltiplos processadores. É composto por três componentes básicos: (1) diretivas de compilação, (2) biblioteca de execução e (3) variáveis de ambiente que alteram o comportamento de um programa.

Considerando que grande parte dos códigos científicos utilizam pelo menos um dos padrões apresentados e que não há suporte nativo para este tipo de aplicação nos ambientes de nuvem atuais, o Nebulous se mostra como uma alternativa para a execução na nuvem de aplicações paralelas desenvolvidas com estas tecnologias.

3 Nebulous Framework

O Nebulous foi projetado para permitir que aplicações paralelas OpenMP e MPI, não desenvolvidas especificamente para nuvens, possam ser executadas neste tipo de ambiente. O *framework* tem o objetivo de automatizar as etapas de implantação, execução e obtenção dos resultados, de modo que o usuário não tenha que interagir diretamente com a nuvem. Outro aspecto que o *framework* adiciona à aplicação é a possibilidade de solicitar mais recursos ou liberar recursos excedentes durante sua execução, aproveitando a escalabilidade do ambiente.

O *framework* proposto é construído sobre a camada do gerenciador de nuvens (ex. OpenNebula, Eucalyptus, Nimbus) e é composto por três componentes básicos: o (1) bloco de descrição de recursos, a (2) API de programação e o (3) Executor, descritos nas Seções 3.1, 3.2 e 3.3, respectivamente. As camadas que compõem a arquitetura do Nebulous são apresentadas na Figura 1.

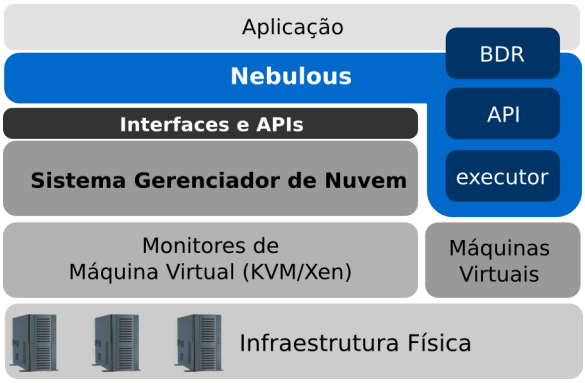

Figura 1. Arquitetura do Nebulous

Atualmente, o Nebulous oferece suporte para as linguagens C e C++.

3.1 Bloco de Descrição de Recursos

O bloco de descrição de recursos (BDR) é utilizado para descrever o ambiente virtual inicial que deverá ser disponibilizado para a aplicação, bem como fornecer as configurações para sua execução.

O BDR deve ser inserido no arquivo do código-fonte C/C++ que contém a função *main*, tendo seu início marcado pela palavra reservada *#neb_config* e seu escopo delimitado por chaves ({}). As configurações são formadas por um conjunto de linhas no formato "campo = valor", onde cada campo está relacionado a uma característica do ambiente virtual ou uma configuração da aplicação. A Figura 2 mostra um exemplo de código-fonte contendo o bloco de descrição.

Como pode-se observar, o bloco é composto por duas seções distintas denotada pelos prefixos *env_* (*environment*) e *app_* (*application*). A seção *env_* define as configurações do ambiente de execução na nuvem, enquanto que a seção *app_* fornece informações para a execução da aplicação. Ambas seções são obrigatórias.

No exemplo, requisita-se um *cluster* composto por 4 máquinas virtuais, cada uma contendo um processador virtual (VCPU) de frequência mínima de 1 GHz (env_min_cpuspeed) e 256 MB de memória RAM

Figura 2. Exemplo de Bloco de Descrição de Recursos

(*env_memory*). Por padrão define-se também uma uma área de *swap* com o dobro do tamanho da memória solicitada.

Além da opção de *clusters*, é possível ainda selecionar ambientes do tipo SMP, correspondente a uma máquina com múltiplos processadores (ou núcleos) ou SINGLE, que corresponde a uma máquina simples com um único processador. Dessa forma é possível executar aplicações sequenciais ou paralelas de memória compartilhada ou distribuída.

As duas linhas seguintes da seção de ambiente descrevem se deverá haver algum disco extra acoplado às MVs (*env_disk*) e qual imagem deverá ser utilizada (*env_image*), respectivamente. A opção por acoplar um disco a uma MV é útil quando há a necessidade de uma movimentar uma grande quantidade de dados para ou de determinada aplicação. Dessa forma, o usuário pode enviar o disco de dados para o repositório da nuvem e utilizá-lo inúmeras vezes para armazenar seus dados. Já a configuração de imagem permite que o usuário envie para a nuvem uma imagem personalizada, contendo todos os requisitos de software para a execução da aplicação.

No caso apresentado na figura, não há a necessidade de nenhum disco acoplado, fato representado pelo valor NONE, nem de uma imagem específica, representada pelo valor ANY . Assim, será utilizada uma imagem padrão para a execução da aplicação, sem a adição de discos.

Por fim, a linha que contém o campo *env_finalize* refere-se à opção de finalizar ou não o ambiente ao final da execução da tarefa e assim, permitindo que os recursos utilizados sejam automaticamente liberados.

A segunda seção do bloco (*app_*) é formada por três configurações de execução: *script* de execução (*app_script*), arquivo de anexos (*app_extras*) e o diretório que armazenará o resultado da aplicação (*app_output*). Note que os anexos devem estar em um arquivo compactado, o qual poderá conter códigos-fonte, bibliotecas, arquivos de entrada e configuração ou qualquer outra dependência necessária para a execução da aplicação.

A Tabela 1 resume os recursos que podem ser configurados no BDR.

Tabela 1. Campos do Bloco de Descrição de Recursos

Seção	Campo	Valores Aceitos	Descrição
env	env_name	[a-zA-Z0-9]+	Nome do ambiente virtual
env	env_type	SINGLE | SMP | CLUSTER	Tipo do ambiente virtual
env	env_nodes	1 – 256	Quantidade de nodos do *cluster*
env	env_VCPU	1 – 256	Quantidade de CPUs virtuais de cada nodo
env	env_min_cpuspeed	[0-9]+	Frequência mínima (em MHz) do processador físico no qual a MV será alocada
env	env_memory	[0-9]+	Quantidade de memória (em MB) disponibilizada para a MV
env	env_disk	[a-zA-Z0-9]+ | NONE	Nome do disco que será acoplado à MV
env	env_image	[a-zA-Z0-9]+ | ANY	Nome da imagem da MV que será inicializada
env	env_finalize	YES | NO	Indica se a MV deverá ser finalizada ou não após o término da execução da aplicação
app	app_script	[a-zA-Z0-9/.]+	Arquivo que define a execução da aplicação
app	app_extras	[a-zA-Z0-9/.]+ | NONE	Arquivo compactado que contém todos os arquivos necessários para a execução da aplicação
app	app_output	[a-zA-Z0-9/.]+	Diretório para o qual serão copiados os resultados da aplicação (local e remoto)

3.2 Nebulous API

Além do BDR, o Nebulous também oferece uma interface de programação (API) pela qual os programadores podem alterar as características do ambiente virtual em tempo de execução. Basicamente, a API fornece quatro primitivas:

#neb_env_nodes(int nodes): Altera a quantidade de nodos do *cluster* para o valor indicado pelo parâmetro *nodes*;

#neb_env_cpu(int cpus, int node): Altera a quantidade de CPUS virtuais para o valor indicado pelo parâmetro *cpus*. A alteração é feita para o nodo indicado no parâmetro *node* e para todos os nodos caso o valor indicado seja menor que zero;

#neb_env_mem(int mem, int node): Altera a quantidade de memória da MV para o valor indicado pelo parâmetro *mem*. A alteração é feita para o nodo indicado no parâmetro *node* e para todos os nodos caso o valor indicado seja menor que zero;

#neb_env_finalize: Finaliza o ambiente de execução instantaneamente. Esta funcionalidade pode ser utilizado em casos de erros ou exceções na aplicação, evitando que os recursos solicitados fiquem ociosos.

A Nebulous API ainda não está totalmente implementada até o presente momento e portanto não foi possível alcançar resultados práticos com o uso da mesma. No entanto, os recursos oferecidos pela API podem ser úteis para as aplicações chamadas maleáveis [6], ou seja, aquelas em que o número de processadores pode ser alterado durante a sua execução. A maleabilidade pode ser interessante em nuvens, já que este tipo de ambiente permite o fornecimento de recursos sob demanda e de modo escalável e elástico. Atualmente, esta característica é bem suportada pelo OpenMP e já apresenta alguns resultados em aplicações desenvolvidas com MPI [8, 2].

Outra situação, na qual a alteração dos recursos em tempo de execução pode ser interessante, é quando existe a cobrança pelos recursos utilizados, como ocorre nas nuvens públicas. Neste caso, a solicitação por mais recursos pode ser feita conforme a demanda da aplicação, reduzindo o custo total da execução.

3.3 Executando Aplicações na Nuvem

O Nebulous foi projetado para que a execução da aplicação na nuvem seja semelhante a uma execução em uma máquina local, tornando desnecessária a interação direta do usuário com o gerenciador de nuvem. Toda e qualquer submissão de tarefa é feita por meio do Executor.

No entanto, antes de submeter uma aplicação para a nuvem, é preciso prepará-la, atendendo alguns pré-requisitos:

1. O usuário deve possuir uma conta na nuvem;

2. O BDR deve ter sido inserido no arquivo de código-fonte C/C++ que contém a função main, conforme descrito na Seção 3.1;

3. No diretório da aplicação devem constar o *script* de compilação (makefile), o arquivo de anexos, o *script* de execução da aplicação e uma chave pública da máquina local.

Após a configuração apropriada, executa-se o Executor passando como parâmetro o arquivo principal da aplicação. Os passos da execução de uma única máquina (SINGLE ou SMP) são mostrados no diagrama da Figura 3 e descritos a seguir.

Inicialmente o Executor faz a análise do BDR, verificando os requisitos do ambiente de execução, e envia o comando para criação da MV para o gerenciador da nuvem (1), que então cria uma instância de MV baseada na imagem solicitada (2). Neste passo, são enviados também um *script* de configuração da MV e a chave pública. O *script* tem a função de personalizar a MV, alterando o nome da máquina, criando um usuário e configurando o acesso via SSH para que seja feito com base na chave recebida.

Após sua inicialização, a MV envia seu IP para o gerenciador de nuvem (3), que repassa a informação para o Executor (4). Na sequência, o Executor envia todos os arquivos necessários para a execução da aplicação na MV (5). Então a aplicação é compilada e executada (6).

Ao final da tarefa, o Executor copia os resultados (7), solicita a finalização do ambiente (8) e termina sua execução. O gerenciador de nuvem então finaliza a MV (9).

É importante salientar que todos os resultados da aplicação devem ser direcionados para arquivos dentro do diretório apontado no BDR, de modo que possam ser corretamente obtidos posteriormente.

A criação de um *cluster* ocorre de forma similar ao apresentado. No entanto, após a criação de todas as máquinas virtuais, um diretório é compartilhado via NFS entre todos os nodos do *cluster* e ocorre a troca de chaves públicas entre eles. Com este procedimento, o *cluster* virtual comporta-se da mesma forma que *cluster* do tipo Beowulf, sendo possível executar as aplicações MPI, que necessitam de acesso confiável entre as máquinas para seu correto funcionamento.

Este compartilhamento de arquivos entre os nodos permite que o Executor copie os arquivos apenas para o nodo 0 do *cluster*, reduzindo o tráfego de dados da rede. A emissão do comando de execução do programa MPI, bem como a coleta dos resultados também são feitos apenas no nodo 0.

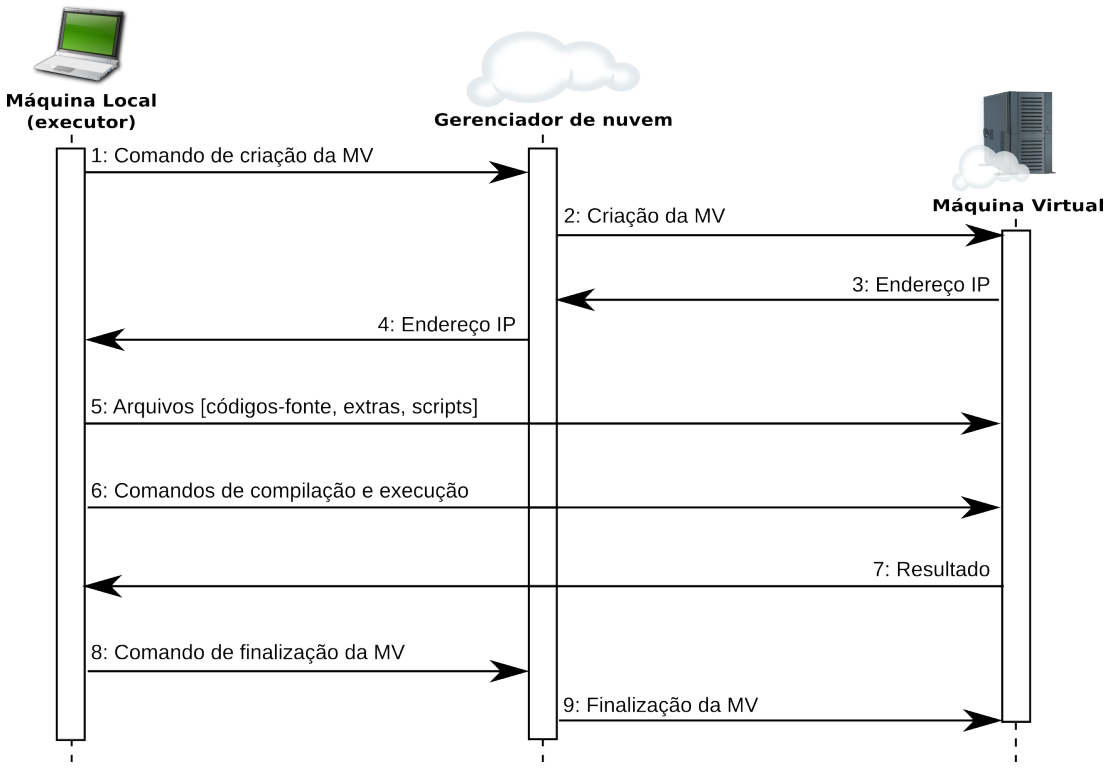

Figura 3. Execução de tarefas no Nebulous

4 Testes e Resultados

Para a execução dos testes, utilizou-se o Nebulous sobre o OpenNebula [21], um ambiente de código aberto para implementação de nuvens computacionais na modalidade IaaS (Infraestrutura como Serviço).

A escolha do OpenNebula foi feita baseada em características e funcionalidades oferecidas. A principal delas é a flexibilidade na criação máquinas virtuais, que permite a configuração dos recursos conforme a demanda da aplicação, ao contrário do que ocorre no Eucalyptus e OpenStack, nos quais deve-se escolher uma classe pré-configurada. Outro ponto relevante considerado foi a existência de uma API para comunicação com gerenciador de nuvem, necessária para o correto funcionamento do *framework*.

A nuvem OpenNebula foi constituída de 1 *front-end* e 4 nodos para a execução de máquinas virtuais, interligadas por Gigabit-Ethernet. A máquina *front-end* é constituída por um processador Intel Xeon dual-core 1.86 GHz, 2 GB de RAM e 400 GB de disco. Os nodos de processamento possuem processador Intel i5 quad-core 3.20 GHz, 3 GB de RAM e 500 GB de disco. Todas as máquinas utilizam sistema operacional Linux Ubuntu 11.04 64 bits.

As máquinas virtuais são criadas a partir de uma imagem KVM no formato qcow2 com sistema operacional Linux Ubuntu Server 11.04 64 bits, com todos os compiladores e bibliotecas necessários instalados de antemão.

Para a validação do *framework* Nebulous, são feitos testes com duas aplicações: um problema de difusão de calor, paralelizado com OpenMP, e uma multiplicação de matrizes para avaliar o comportamento de uma aplicação paralelizada com MPI.

4.1 Difusão de Calor - OpenMP

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 [15].

A solução consiste na montagem de um sistema de equações e sua resolução (via Gradiente Conjugado) a cada passo de tempo. A aplicação foi implementada em C e OpenMP. Para a medição do desempenho, simulou-se 100 passos de tempo em domínios com 1024×1024 e 2048×2048 células, utilizando 1, 2 e 4 *threads*.

A aplicação foi configurada de acordo com o mostrado na Seção 3, sendo que a única modificação necessária foi a inclusão do BDR, mostrado a seguir.

```
#neb_config
{
    env_name=HeatTransfer
    env_type=SMP
    env_VCPU=4
    env_min_cpuspeed=1000
    env_memory=1024

    app_script=script.sh
    app_output_dir=output
}
```

A máquina virtual solicitada é do tipo SMP e é composta por 4 núcleos de processamento com 1 GB de memória. Na sequência, foi feita a submissão da tarefa via Executor, que encarregou-se da geração do ambiente descrito no bloco e também da execução e recuperação dos resultados.

A Figura 4 mostra os tempos de execução da aplicação executada na nuvem comparados com os tempos obtidos em uma máquina física de mesma configuração dos nodos da nuvem.

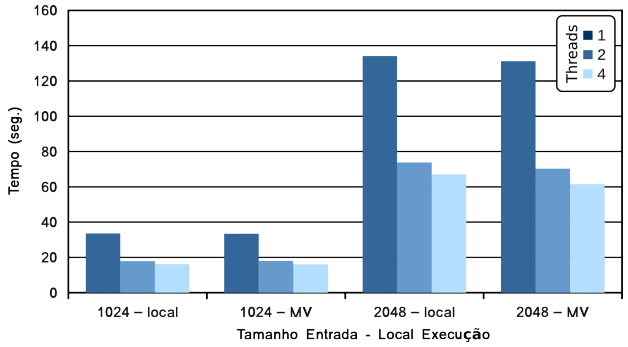

Figura 4. Tempos de Execução - OpenMP

Observa-se que os tempos obtidos na nuvem (*MV*) são bastante semelhantes máquina física (*local*), não apresentando sobrecarga (*overhead*) significativa. Nos gráficos não estão inclusos o tempo de criação e configuração da máquina virtual, que neste teste foi aproximadamente 2:30 minutos.

A Figura 5 apresenta o resultado de uma simulação de 180 iterações em um domínio com 100x100 células, com temperaturas iniciais de 10^oC nas bordas e 1^oC no interior do domínio. Os dados foram obtidos utilizando 2 *threads*.

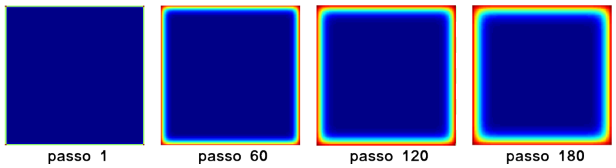

Figura 5. Resultado da aplicação

O resultado mostra-se correto e de acordo com o esperado, com as maiores temperaturas fluindo para as áreas de menor temperatura.

4.2 Multiplicação de Matrizes - MPI

A multiplicação de matrizes é uma operação fundamental em muitas aplicações numéricas de álgebra linear. Sua implementação eficiente em computadores paralelos é uma questão de primordial importância para bibliotecas de software científico.

A aplicação foi implementado em C, utilizando a biblioteca OpenMPI [10] para a troca de mensagens. As operações foram testadas com arquivos de entrada contendo matrizes de dimensão 2048×2048.

Foram utilizadas três configurações de ambiente de execução: (1) duas máquinas físicas com quatro núcleos cada, configuradas manualmente (4x2-MF); (2) *cluster* virtual de oito nodos com um único processador (8x1-MV), criado automaticamente pelo Nebulous; e um *cluster* virtual de quatro nodos com dois núcleos cada (4x2-MV), também criado pelo Nebulous.

É importante salientar que a única configuração necessária foi a inserção do BDR no código-fonte. Toda a configuração do *cluster* virtual, que envolve a criação das diversas máquinas virtuais e a troca de certificados, foi feita automaticamente pelo *framework*, assim como a toda a etapa de execução. O BDR utilizado na configuração do ambiente 8x1-MV pode ser visto a seguir.

```
#neb_config
{
    env_name=MM
    env_type=CLUSTER
    env_nodes=8
    env_VCPU=1
    env_min_cpuspeed=1000
    env_memory=512

    app_script=script.sh
    app_extras=matrix.tar.gz
    app_output_dir=output
}
```

Os resultados, apresentados na Figura 6 mostram que as aplicações MPI executadas em ambientes virtuais sofrem um impacto maior no desempenho do que as aplicações OpenMP, com aumento no tempo de execução variando entre 4% e 40%. Os maiores quedas de rendimento ocorreram nas execuções utilizando oito processos, justamente onde a sobrecarga resultante da latência da comunicação é mais significativa.

Estes resultados já eram esperados e estão de acordo com alguns resultados publicados na literatura [9, 13]. Novamente, nos tempos exibidos nos gráficos não estão adici-

onados o custo para a criação do *cluster* virtual, que para estes testes foram de aproximadamente 4 minutos.

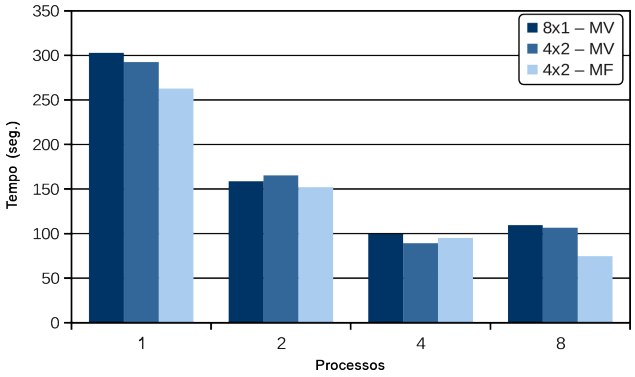

Figura 6. Tempos de Execução - MPI

Considerando os resultados obtidos nos testes envolvendo as duas aplicações, o *framework* comportou-se dentro do esperado, criando os ambientes virtuais (SMP e CLUSTER) de acordo com o especificado, e automatizando com sucesso as tarefas de execução e recuperação dos resultados de modo simples e uniforme.

É importante destacar que os tempos para a criação dos ambientes não relacionam-se diretamente com os tempos dos experimentos. O tempo de criação é basicamente influenciado pela imagem da máquina virtual utilizada e pela quantidade de nodos utilizados, no caso de um *cluster*. A simplificação da imagem, optando por versões mais enxutas de sistema operacional pode contribuir significativamente para tornar esse processo mais rápido.

5 Trabalhos Relacionados

Recentemente, muitos trabalhos têm explorado a migração de aplicações científicas para ambientes de nuvem. Jackson et al (2010) compara o desempenho de oito diferentes aplicações (CAM, Gamess, GTC, IMPACT-T, MAESTRO, MILC, Paratec e HPCC) obtidos em dois aglomerados virtuais privados e na nuvem Amazon EC2. Em trabalho semelhante, Evangelinos e Hill (2008) apresentam a execução de um modelo atmosférico na nuvem EC2 e analisam o impacto do uso de diferentes implementações de bibliotecas do padrão MPI no ambiente virtual.

Ainda no contexto de aplicações científicas, alguns autores abordaram a execução de *workflows* em ambientes de nuvem [12], [14] e [23]. Nestes trabalhos, a automatização da execução das tarefas é feita pelos gerenciadores de *workflow*, tal como Pegasus [5] e Kepler [16] acoplados a algum sistema de nuvem.

Apesar do uso da computação em nuvem estar tomando maiores proporções no cenário científico, não há pesquisas publicadas abordando o suporte à execução de aplicações paralelas OpenMP e MPI em nuvens, como é o caso do Nebulous. Os trabalhos que mais se assemelham à presente proposta, em algum aspecto específico, são a Neptune [1] a DADL [17] e o Nimbus Context Broker [18].

A Neptune é uma linguagem de domínio específico, baseada em Ruby, criada para a execução de aplicações de alto desempenho em nuvens AppScale. Seu funcionamento é semelhante a um *script* de execução, onde configura-se o tipo da aplicação (MPI, X10, Mapreduce), qual é o código que será executado e o número de máquinas que será utilizado.

A DADL (*Distributed Application Description Language*) é uma linguagem para a especificação da arquitetura, comportamento e requisitos de hardware de uma aplicação distribuída. A linguagem será utilizada em um *framework* de gerenciamento e configuração de software distribuído.

Por fim, o Nimbus Context Broker é uma ferramenta para transformar um grupo de máquinas virtuais alocadas individualmente em um *cluster*, bastante semelhante ao proposto pelo Nebulous.

6 Conclusão e Trabalhos Futuros

A principal contribuição deste trabalho foi o desenvolvimento e a apresentação do *framework* Nebulous, cujo objetivo é facilitar a implantação e a execução de aplicações científicas paralelas em nuvens computacionais.

De acordo com os resultados dos testes de validação, o Nebulous permitiu a implantação das aplicações em ambientes virtuais com diversas configurações de modo simples, uniforme e escalável e com o mínimo esforço, cumprindo com os objetivos de projeto, que era o de fornecer suporte adequado a tarefas OpenMP e MPI na nuvem.

Apesar do *framework* já estar em funcionamento, existem algumas propostas de trabalho futuro para a melhora do mesmo. A primeira delas é a implementação completa da API, de modo a possibilitar a aquisição ou liberação recursos em tempo de execução, conforme descrito na Seção 3.2. As demais envolvem o tratamento de erros e exceções, redução do tempo de criação dos ambientes virtuais, e a otimização da localidade dos nodos de um *cluster* virtual, de modo a reduzir a latência de comunicação.

7 Agradecimentos

Os autores gostariam de agradecer à CAPES pelo financiamento parcial deste trabalho.

Referências

[1] C. Bunch, N. Chohan, C. Krintz, and K. Shams. Neptune: a domain specific language for deploying hpc software on cloud platforms. In *Proceedings of the 2nd international workshop on Scientific cloud computing*, ScienceCloud '11, pages 59–68, New York, NY, USA, 2011. ACM.

[2] M. Cera, Y. Georgiou, O. Richard, N. Maillard, and P. Navaux. Supporting malleability in parallel architectures with dynamic cpusetsmapping and dynamic mpi. In K. Kant, S. Pemmaraju, K. Sivalingam, and J. Wu, editors, *Distributed Computing and Networking*, volume 5935 of *Lecture Notes in Computer Science*, pages 242–257. Springer Berlin / Heidelberg, 2010.

[3] B. Chapman, G. Jost, and R. van der Pas. *Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation)*. The MIT Press, 2007.

[4] D. de Oliveira and E. Ogasawara. Is cloud computing the solution for brazilian researchers? *International Journal of Computer Applications*, 6(8):19–23, September 2010. Published By Foundation of Computer Science.

[5] E. Deelman, M. Livny, G. Mehta, and A. Pavlo. Pegasus and dagman from concept to execution: Mapping scientific workflows onto todays cyberinfrastructure. *IOS*, 2008.

[6] T. Desell, K. E. Maghraoui, and C. A. Varela. Malleable applications for scalable high performance computing. *Cluster Computing*, 10:323–337, September 2007.

[7] T. Deshane, Z. Shepherd, J. Matthews, M. Ben-Yehuda, A. Shah, and B. Rao. Quantitative comparison of Xen and KVM. In *Xen summit*, Berkeley, CA, USA, June 2008. USENIX association.

[8] K. El Maghraoui, T. J. Desell, B. K. Szymanski, and C. A. Varela. Malleable iterative mpi applications. *Concurrency and Computation: Practice and Experience*, 21(3):393–413, 2009.

[9] C. Evangelinos and C. N. Hill. Cloud computing for parallel scientific hpc applications: Feasibility of running coupled atmosphere-ocean climate models on amazon â s ec2 . *October*, 2(2.40):2–34, 2008.

[10] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, September 2004.

[11] W. Gropp, E. Lusk, and R. Thakur. *Using MPI-2: Advanced Features of the Message-Passing Interface*. MIT Press, Cambridge, MA, USA, 1999.

[12] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, and J. Good. On the use of cloud computing for scientific workflows. In *Proceedings of the 2008 Fourth IEEE International Conference on eScience*, pages 640–645, Washington, DC, USA, 2008. IEEE.

[13] K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, and N. J. Wright. Performance analysis of high performance computing applications on the amazon web services cloud. In *CloudCom'10*, pages 159–168, 2010.

[14] G. Juve and E. Deelman. Scientific workflows and clouds. *ACM Crossroads*, pages 14–18, 2010.

[15] J. H. Lienhard and J. H. Lienhard. *A Heat Transfer Textbook - 3rd ed.* Phlogiston Press: Cambridge, Massachusetts, 2008.

[16] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the kepler system: Research articles. *Concurr. Comput. : Pract. Exper.*, 18:1039–1065, August 2006.

[17] J. Mirkovic, T. Faber., P. Hsieh, G. Malayandisamu, and R. Malavia. Dadl: Distributed application description language. Technical Report ISI-TR-664, USC/ISI, 2010.

[18] N. Project. `http://www.nimbusproject.org/`.

[19] P. Sempolinski and D. Thain. A comparison and critique of eucalyptus, opennebula and nimbus. *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, November:417–426, 2010.

[20] Y. Simmhan, C. Van Ingen, G. Subramanian, and J. Li. Bridging the gap between desktop and the cloud for escience applications. *2010 IEEE 3rd International Conference on Cloud Computing*, pages 474–481, 2010.

[21] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster. Virtual infrastructure management in private and hybrid clouds. *IEEE Internet Computing*, 13:14–22, 2009.

[22] H.-L. Truong and S. Dustdar. Cloud computing for small research groups in computational science and engineering: current status and outlook. *Computing*, 91:75–91, January 2011.

[23] J.-S. Vöckler, G. Juve, E. Deelman, M. Rynge, and B. Berriman. Experiences using cloud computing for a scientific workflow application. In *Proceedings of the 2nd international workshop on Scientific cloud computing*, ScienceCloud '11, pages 15–24, New York, NY, USA, 2011. ACM.

[24] L. Wang, J. Tao, M. Kunze, A. C. Castellanos, D. Kramer, and W. Karl. Scientific cloud computing: Early definition and experience. In *Proceedings of the 2008 10th IEEE International Conference on High Performance Computing and Communications*, HPCC '08, pages 825–830, Washington, DC, USA, 2008. IEEE Computer Society.