Hermes: uma ferramenta de entrega serverless para GPUs

Tiago Martins Nápoli

Proposta de Trabalho de Conclusão de Curso
apresentado à disciplina
MAC0499
(Trabalho de Formatura Supervisionado)

Orientador: Prof. Dr. Alfredo Goldman
Co-Orientador: Renato Cordeiro Ferreira

São Paulo, Abril de 2019

Conteúdo

Capítulo 1

Introdução

Capítulo 2

Revisão de Literatura

2.1 Virtualização

Virtualização é a criação de uma versão virtual - baseada em software - de algum recurso, como servidores, dispositivos de armazenamento, rede ou sistemas operacionais. A partir do uso desta técnica é possível aumentar o aproveitamento de elementos de hardware, facilitar seu gerenciamento, além de diminuir os custos: a partir de um recurso de hardware é possível, por meio da virtualização, ter várias versões virtuais dele, eliminando em diversos casos a necessidade de compra de mais hardware. A seguir serão apresentados duas tecnologias muito importantes nos dias de hoje, que utilizam técnicas de virtualização: as Máquinas Virtuais e os Containers.

2.1.1 Máquinas Virtuais

Uma Máquina Virtual (*Virtual Machine* - VM) pode ser definida como um software que fornece uma "duplicata eficiente e isolada de uma máquina real"[1]. A partir do uso das VMs é possível criar, em um único servidor ou máquina, diversos ambientes de execução isolados, ao invés de um único, que seria o padrão sem o uso de virtualização. As vantagens disso são muitas e serão discutidas mais a frente, antes é necessário uma contextualização do funcionamento das VMs.

As Máquinas Virtuais são fundamentadas no chamado *Virtual Machine Monitor*, ou *hypervisor*. Ele é o software responsável por hospedar as diversas VMs que podem estar sendo executadas na mesma máquina, atuando como uma camada intermediária entre elas e o hardware da máquina real. Nesse contexto, as VMs hospedadas pelo *hypervisor* são chamadas *guests*, e a máquina real sobre a qual o *hypervisor* executa é chamada *host*.

Para hospedar as VMs, o *hypervisor* virtualiza todos os recursos de hardware do *host* (*e.g.*, processadores, memória, disco, entre outros), fornecendo uma interface para utilização deles. O *hypervisor* também controla a alocação de recursos do *host* para os *guests*, e é responsável por escalonar cada VM de maneira similar a como o SO escalona processos [2]. Além disso, e essa característica é crucial para as aplicações atuais das VMs, o *hypervisor* é responsável por garantir o isolamento entre cada VM hospedada [3].

Essas características do *hypervisor* são as principais responsáveis pelas vantagens que as VMs trazem consigo. Algumas delas estão listadas as seguir [2]:

a) **Encapsulamento**: Como as VMs em uma mesma máquina têm isolamento garantido entre elas, os ambientes de execução de cada uma podem ter configuração totalmente diferentes

da outra, inclusive sistemas operacionais distintos. Isso permite a empresas e desenvolvedores executar cada serviço ou aplicação em uma VM diferente, cada uma com a configuração adequada para o trabalho a ser realizado [4, p. 17].

b) **Segurança, Confiabilidade e Disponibilidade**: O isolamento a nível de hardware trazido pelo *hypervisor* garante que vulnerabilidades ou falhas em aplicações de alguma VM não afete outras VMs na mesma máquina.

c) **Custo**: Historicamente existia uma prática de executar somente uma aplicação ou serviço em cada servidor, a fim de evitar que a execução de um interferisse em outro. Há algumas décadas essa prática, sumarizada pela frase *"one server, one application"*, culminou com *data centers* abarrotados de servidores, com altos custos de refrigeração, pouco espaço físico restante e máquinas com grande parte do poder de processamento ocioso. Com a introdução das máquinas virtuais, que mantém entre si o isolamento desejado, um único servidor com várias VMs passou a fazer o trabalho que vários servidores faziam. Isso possibilitou que os *data centers* tivessem menos máquinas reais, cada uma com menos tempo de ociosidade, reduzindo os custos de manutenção, refrigeração e energia [4, p. 3-14].

d) **Adaptabilidade à variação na carga de trabalho computacional**: Essa variação pode ser tratado deslocando a alocação de recursos computacionais de uma VM à outra. Existem soluções automáticas que fazem essa alocação de recursos dinamicamente [5].

2.1.2 Containers

Um Container pode ser definido como um grupo de processos sobre o qual é aplicado uma camada de isolamento entre ele e o resto do sistema. Este isolamento é feito de modo que o container tenha um sistema de arquivos privado e os processos em seu interior sejam limitados a detectar e utilizar somente recursos do sistema que tenham sido atribuídos ao container [6, p. 916]. Os processos conteinerizados utilizam o Kernel e outros serviços (e.g. drivers) do SO em que o container foi criado, retirando a necessidade dos containers armazenarem um SO completo em seus sistemas de arquivos.

Deste modo, os Containers são capazes de criar ambientes de execução isolados, assim como as Máquinas Virtuais, trazendo consigo os inúmeros benefícios que estas oferecem: segurança, encapsulamento, confiabilidade, disponibilidade, dentre outras. Há, entretanto, diferenças cruciais nos modelos de cada um, o que implica diversas vantagens e desvantagens de um em relação ao outro, havendo a necessidade de um comparativo entre os dois:

a) **Performance**: Quando à performance os containers têm vantagem. Pelo fato das VMs dependerem de um *hypervisor*, uma camada a mais entre as instruções dos processos e a execução pelo hardware, há um atraso inerente a todas as instruções de hardware realizadas dentro das VMs. Com os Containers isso não ocorre, os processos se comunicam diretamente com o Kernel e atrasos adicionais são pequenos, comparados às VMs [7].

b) **Inicialização**: As VMs, por serem um SO completo, necessitam de um processo de *boot*, algo que os containers não necessitam, já que utilizam os recursos e kernel do sistema que os hospeda [6, p. 904-906], por isso o tempo de inicialização dos containers é muito inferior [8].

c) **Espaço em Disco**: Mais uma vez por serem um SO completo, as VMs utilizam muito mais espaço em disco se comparadas aos Containers. Estes necessitam armazenar em seu sistema de arquivos somente as dependências da aplicação que encapsulam [6, p. 904-906].

d) **Segurança**: Nesse aspecto as VMs têm vantagem. No caso dos Containers, todos as instâncias que estiverem ativas em um SO compartilham do mesmo Kernel e recursos computacionais. No caso de haver uma falha nesse pontos, todos os containers ativos são afetados. As VMs estão livres dessa vulnerabilidade [8].

e) **Isolamento**: Por virtualizarem em nível de hardware, as VMs têm um isolamento superior. O isolamento realizado pelos container é em nível de processo, havendo compartilhamento do Kernel, por exemplo, como já mencionado [6, p. 904-906].

f) **Flexibilidade quanto à SO**: Em uma mesma máquina pode-se desejar virtualizar vários SOs diferentes, como Windows, Linux, ou MacOS. Isso não é realizável somente com Containers. Como eles não carregam consigo um SO, e dependem do Kernel do SO que os hospeda, não seria possível executar um container com uma aplicação de certo SO se este for executado em outro SO diferente. Este não é um problema que as VMs enfrentam [6, p. 904-906].

2.2 Nuvem

A Nuvem (*Cloud*) em essência pode ser definida como um grande conjunto de recursos virtualizados (*e.g.* hardware, plataformas de desenvolvimento, serviços) com fácil acesso e usabilidade [9]. Estes recursos podem ser dinamicamente reconfigurados e ajustados segundo as variação da necessidade de uso, garantindo o que é denominado elasticidade: a alteração na demanda por certo recurso desencadeia ajustes em seu aprovisionamento, garantindo uso otimizado[9]. Tipicamente, a utilização da *cloud* é cobrada segundo um modelo denominado *pay-per-use*, em que os usuários pagam somente por quanto e o que usarem, segundo alguma métrica criada pelo provedor de *cloud* [9]. Neste modelo de cobrança, os usuários são protegidos pela SLA (Contrato de Nível de Serviço), que descreve os compromissos do provedor quanto à disponibilidade e tempo de atividade dos recursos oferecidos.

Neste contexto, surgiu também o termo Computação em Nuvem (*Cloud Computing*), que se refere à utilização dos recursos oferecidos por um provedor de *cloud* [10]. Esta utilização e o gerenciamento dos recursos é feita, pelo cliente, por meio de uma rede, geralmente a Internet.

Atualmente a Nuvem assume um papel extremamente importante para a indústria e tem atraído inclusive a comunidade científica, dependendo do balanço de custos [11]. Este projeto, seguindo essa tendência, utilizará vários conceitos criados e popularizados em função do *Cloud Computing*, e esta seção oferecerá contexto e definições quanto a esses conceitos.

2.2.1 Serviços da Nuvem

Os diversos serviços oferecidos pelos provedores de Nuvem podem ser agrupados, tradicionalmente, nas seguintes categorias:

a) Infraestrutura como Serviço (IaaS): Nesse modelo tipicamente são oferecidos recursos de hardware e servidores virtualizados. Os clientes, ao optarem por esse serviço, são cobrados somente

pelos recursos consumidos e não precisam se preocupar com compra, instalação e manutenção de hardware e servidores em um *data center*, mas sim com a implantação de seus softwares utilizando os recursos virtualizados disponibilizados (*e.g.* VMs) e o gerenciamento de tais recursos. Ao provedor de nuvem é depositada a responsabilidade de garantir a disponibilidade de recursos de hardware, servidores e poder computacional, para que os clientes possam escalar suas aplicações facilmente [12].

b) Plataforma como Serviço (PaaS): Esse serviço consiste no provedor fornecer uma infraestrutura de software sobre a qual os clientes podem desenvolver e lançar determinadas classes de aplicações e serviços. A plataforma oferecida é baseada na infraestrutura do provedor, e sua utilização é cobrada seguindo o modelo *pay-per-use*, segundo a utilização de recursos computacionais pelo cliente. Apesar disso, a infraestrutura é abstraída da plataforma, ou seja, ao cliente não é necessário gerenciar recursos virtualizados: seu foco é o produto que esteja desenvolvendo [12].

c) Software como Serviço (Saas): Nesta modalidade, aplicações completas com funcionamento baseado na infraestrutura de um provedor de nuvem são oferecidas ao usuário final. Os clientes do serviço não precisam se preocupar com gerenciamento de infraestrutura ou da plataforma para garantir seu funcionamento. Um exemplo comum aplicação que segue esse modelo são os WebMails [12].

2.2.2 Arquitetura *Serverless*

A Arquitetura *Serverless* refere-se a um padrão de desenvolvimento de software em que funções sem estado são utilizadas para compor uma aplicação ou parte dela. Essas funções são executadas na nuvem, sem a necessidade de gerenciamento de infraestrutura por parte do desenvolvedor [13]: o provedor de nuvem é responsável pela execução, monitoramento, manutenção e escalabilidade da aplicação.

Nesse modelo, a cobrança pelo serviço é baseada no tempo de execução das funções. Isso é uma grande vantagem pelo fato de que, nessa arquitetura, uma função pode ser escalado a zero, ou seja, se não estiver em uso, não será executada [13]. Essas características permitem ao cliente pagar somente pelo tempo que as funções foram de fato usadas.

Por fim, outra característica chave das Arquiteturas *Serverless* é a orientação a eventos [13]: a execução de funções é feita em resposta a eventos predefinidos.

2.3 GPUs

As GPUs (Unidades de Processamento Gráfico) são processadores que foram criados inicialmente para lidar com operações de renderização de modo mais eficiente. Apesar de possuírem em princípio essa finalidade, a arquitetura altamente paralela que possuem fomentou o interesse no uso de GPUs em aplicações mais gerais, originando o que foi chamado de GPGPU (Computação de Propósito Geral em GPU). Em virtude do imenso desenvolvimento da arquitetura das GPUs e das práticas de GPGPU desde a sua criação, hoje a importância delas para a comunidade científica e a indústria é enorme: aplicações aceleradas por GPUs são usadas por áreas como Inteligência

Artificial, Aprendizado de Máquina, Física Computacional, Química Computacional, Exploração de Petróleo e Gás Natural, Astrofísica, dentre muitas outras [14].

Capítulo 3

Proposta

O objetivo deste projeto é a criação de Hermes - uma ferramenta de código aberto que oferecerá uma experiência *serverless* de desenvolvimento em GPUs. Definindo e implementando uma API, Hermes oferecerá a possibilidade de compilar e executar códigos para GPUs em servidores remotos, sem a necessidade de configurações ou instalações no ambiente de execução destes servidores.

Hermes visa eliminar um dos grandes inconvenientes no desenvolvimento para GPUs: a configuração do ambiente de desenvolvimento. Hoje, as opções mais comuns são o desenvolvimento local ou o remoto, em máquinas virtuais, ambos trazendo inconvenientes para compilar e executar código: A alternativa local exige a disponibilidade de uma GPU por programador e configuração de drivers e kits de desenvolvimento; a alternativa remota exige acesso via SSH, alterações no código com editores de linha de comando e testes de maneira manual.

Já existem serviços proprietários que resolvem algo semelhante ao que Hermes está se propondo a solucionar: *Spell*[15] e *Paperspace - Gradient*[16], por exemplo. Essas ferramentas, entretanto, são centradas apenas em desenvolvimento para Inteligência Artificial e Aprendizado de Máquina, ao passo que Hermes terá foco no uso geral de GPUs em ambiente remoto. Apesar disso, elas validam a existência de demanda para o problema que originou a ideia para o Hermes.

Semelhantemente aos produtos mencionados, Hermes implementará uma API para execução remota de código, e a disponibilizará por meio de uma CLI. O *backend* do serviço será implantado em um conjunto de servidores e terá como fluxo de trabalho base o seguinte:

a) Receber requisições de execução de código, juntamente com os inputs.

b) Empacotar os arquivos fonte necessários e as dependências em um container.

c) Escalonar a execução do container em um servidor com GPU.

d) Executar o container e ao final disponibilizar o *output* ao usuário.

A princípio a implantação e testes serão realizados em um servidor localizado no IME, o Ratel, que possui uma *GTX1080 Ti*, GPU da NVIDIA.

Por fim, como prova de conceito do sistema, planeja-se criar, utilizando o Hermes como ferramenta, uma plataforma de submissões para a Maratona de Programação Paralela. Esta nova plataforma substituirá a versão atual que possui uma interface relativamente antiga e lida com GPUs de forma não ideal.

Capítulo 4

Cronograma de Atividades

O projeto foi dividido em cinco fases:

1) **Planejamento Arquitetural**: A primeira fase envolverá a definição da arquitetura do Hermes e será um processo iterativo entre, a princípio, as seguintes atividades:

   a) Definição de funcionalidades do sistema.

   b) Planejamento e aprimoramento da arquitetura básica do *backend*.

   c) Estudo de ferramentas livres existentes que possam auxiliar na implantação do sistema: As atividades realizadas pelo *backend* envolverão empacotamento de código em containers (seguindo um paradigma *serverless*), orquestração de containers, escalonamento de execução de containers em máquinas virtuais, monitoramento de execução, dentre outros problemas já estudados pela comunidade científica e pela indústria. Hoje existem ferramentas que auxiliam a resolução destes problemas e nesta etapa será estudada a viabilidade do uso de ferramentas encontradas.

   Ao fim dessa fase espera-se existir uma arquitetura do sistema com módulos bem definidos.

2) **Implementação da API**: A segunda fase será a implementação e testes dos módulos do sistema, além de sua implantação no Ratel. Após isso a API estará funcional.

3) **Implementação da CLI**: A terceira fase do projeto será a implementação e testes da CLI do Hermes.

4) **Implementação da Prova de Conceito**: A quarta fase será a implementação da prova de conceito do sistema, a plataforma de submissão para a Maratona de Programação Paralela. Esta fase envolverá estudo do sistema já existente, definição das principais funcionalidades e, por fim, implementação do *backend* (utilizando o Hermes) e *frontend*.

5) **Monografia**

O cronograma aproximado para o projeto é o seguinte:

	abr	mai	jun	jul	ago	set	out
1ª fase - Planejamento Arquitetural	X	X					
2ª fase - Implementação da API		X	X				
3ª fase - Implementação da CLI			X	X			
4ª fase - Implementação da Prova de Conceito				X	X		
5ª fase - Monografia					X	X	X

Bibliografia

[1] G. J. Popek and R. P. Goldberg, "Formal requirements for virtualizable third generation architectures," **Communications of the ACM**, vol. 17, no. 7, pp. 412–421, 1974. 2

[2] D. A. Menascé, "Virtualization: Concepts, applications, and performance modeling," in **31th International Computer Measurement Group Conference, December 4-9, 2005, Orlando, Florida, USA, Proceedings**, pp. 407–414, 2005. 2

[3] L. P. Lotti and A. B. Prado, "Sistemas virtualizados – uma visão geral," 2010. 2

[4] M. Portnoy, **Virtualization Essentials**. Wiley, 1st ed., 2012. 3

[5] M. N. Bennani and D. A. Menasce, "Resource allocation for autonomic data centers using analytic performance models," in **Second International Conference on Autonomic Computing (ICAC'05)**, pp. 229–240, June 2005. 3

[6] E. Nemeth, G. Snyder, T. R. Hein, B. Whaley, and D. Mackin, **UNIX and Linux System Administration Handbook (5th Edition)**. Addison-Wesley Professional, 5th ed., 2017. 3, 4

[7] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and linux containers," in **2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)**, pp. 171–172, March 2015. 3

[8] K. Agarwal, B. Jain, and D. E. Porter, "Containing the hype," in **Proceedings of the 6th Asia-Pacific Workshop on Systems**, APSys '15, (New York, NY, USA), pp. 8:1–8:9, ACM, 2015. 3, 4

[9] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: Towards a cloud definition," **SIGCOMM Comput. Commun. Rev.**, vol. 39, pp. 50–55, Dec. 2008. 4

[10] V. Rajaraman, "Cloud computing," **Resonance**, vol. 19, pp. 242–258, Mar 2014. 4

[11] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The cost of doing science on the cloud: The montage example," in **SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing**, pp. 1–12, Nov 2008. 4

[12] C. N. Höfer and G. Karagiannis, "Cloud computing services: taxonomy and comparison," **Journal of Internet Services and Applications**, vol. 2, pp. 81–94, Sep 2011. 5

[13] I. Baldini, P. C. Castro, K. S. Chang, P. Cheng, S. J. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. M. Rabbah, A. Slominski, and P. Suter, "Serverless computing: Current trends and open problems," **CoRR**, vol. abs/1706.03178, 2017. 5

[14] NVIDIA, "Gpu-accelerated applications," 2019. 6

[15] Spell, "Deep learning and ai development platform." 7

[16] Gradient, "A suite of tools for exploring data, training neural networks, and running gpu compute jobs." 7