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

Nos últimos anos, as GPUs têm ganhado cada vez mais importância na comunidade científica e na indústria. Sua arquitetura altamente paralela mostrou-se efetiva **?** em acelerar diversos tipos de aplicações de grande interesse, como aplicações de Aprendizado de Máquina, Inteligência Artificial, Cálculo Numérico, dentre muitas outras **?**. Essa crescente importância das GPUs tem incentivado inovações na maneira como utilizá-las e provisioná-las: Os principais provedores de nuvem hoje oferecem máquinas virtuais com acesso à GPU; *startups* foram fundadas com produtos que tentam facilitar o fluxo de trabalho com GPUs para certos usos **?**; *frameworks* de certas linguagens passaram a oferecer suporte à aceleração com GPUs **?**, dentre outros exemplos.

Apesar dessas inovações, o fluxo de desenvolvimento de aplicações gerais para GPUs ainda tem certos obstáculos. Um desses é a configuração do ambiente de desenvolvimento: tanto o desenvolvimento local quanto o remoto trazem consigo inconvenientes para compilar e executar código. A alternativa local exige disponibilidade de uma GPU por programador, 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.

Dado esse inconveniente, a proposta desse projeto é criar o Hermes, uma ferramenta de código aberto com o objetivo de eliminar grande parte dessas dificuldades, mantendo certas vantagens tanto do desenvolvimento local quanto do remoto, como o acesso a GPUs mais poderosas, uma vantagem comum do desenvolvimento remoto, e o uso do ambiente local, com o qual o desenvolvedor está confortável, uma vantagem do desenvolvimento local.

Neste documento será descrito a proposta de desenvolvimento do Hermes: O capítulo 2 faz uma revisão de literatura de certos conceitos importantes para o projeto. Os capítulo 3 e 4 descrevem, respectivamente, a proposta do Hermes e o cronograma de atividades para desenvolvimento.

Capítulo 2

Revisão de Literatura

Para compreender o trabalho a ser realizado neste projeto, há três conceitos essenciais que serão utilizados: virtualização, nuvem e GPUs. Este capítulo apresenta os principais conhecimentos necessários sobre esses assuntos que serão posteriormente utilizados no detalhamento da proposta apresentada no capítulo 3.

2.1 Virtualização

Virtualização é a criação de uma versão virtual – baseada em software – de algum recurso, tais como servidores, dispositivos de armazenamento, redes ou sistemas operacionais. A partir do uso desta técnica é possível aumentar o aproveitamento de elementos de hardware, facilitar seu gerenciamento e diminuir 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 apresentadas duas tecnologias muito importantes nos dias de hoje, que utilizam técnicas de virtualização: as máquinas virtuais e os contêineres.

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 **?**. A partir do uso das VMs é possível criar, em uma única máquina, diversos ambientes de execução isolados ao invés de um único, que seria o padrão sem o uso de virtualização.

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 real. Nesse contexto, as VMs hospedadas pelo *hypervisor* são chamadas **hóspedes** (***guests***), e a máquina real sobre a qual o *hypervisor* executa é chamada **hospedeiro** (***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 a utilização dos mesmos. 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 (Sistema Operacional) escalona processos **?**. Além disso, e essa característica é um dos grandes atrativos das VMs, o *hypervisor* é responsável por garantir o isolamento entre cada VM hospedada **?**.

Este isolamento garantido pelo *hypervisor* é um dos grandes responsáveis pelas vantagens que as VMs trazem consigo. Algumas delas estão listadas as seguir **?**:

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 executarem suas aplicações em VMs diferentes, cada uma com a configuração adequada para o trabalho a ser realizado [**?**, 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**: As VMs eliminam a necessidade de possuir um servidor exclusivo para cada aplicação, algo que era uma prática comum há algumas décadas, para evitar que a execução de uma aplicação interferisse em outras. Como um único servidor pode executar várias VMs e elas têm isolamento entre si, a carga de trabalho de vários servidores é condensado em um só, garantindo economia nos custos de manutenção, refrigeração e energia, além da diminuição do tempo ocioso dos servidores [**?**, p. 3-14].

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

2.1.2 Contêineres

Um Contêiner pode ser definido como um grupo de processos sobre o qual é aplicado uma camada de isolamento entre eles e o resto do sistema. Este isolamento é feito de modo que o contêiner 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 contêiner [**?**, p. 916]. Os processos conteinerizados reutilizam o *kernel* e outros serviços (e.g., drivers) do SO em que o contêiner foi criado, retirando a necessidade dos contêineres armazenarem um SO completo em seus sistemas de arquivos.

Deste modo, os contêineres 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 cria a necessidade de um comparativo entre as vantagens e desvantagens dos dois:

a) **Performance**: 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 contêineres isso não ocorre, os processos se comunicam diretamente com o *kernel* e atrasos adicionais são pequenos, comparados às VMs **?**.

b) **Inicialização**: As VMs, por terem um SO completo, necessitam de um processo de inicialização (*boot*), algo que os contêineres não necessitam, já que utilizam os recursos e *kernel* do

sistema que os hospeda [?, p. 904-906]. Por isso, o tempo de inicialização dos contêineres é muito inferior ?.

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

d) **Segurança**: No caso dos contêineres, todas as instâncias que estiverem ativas em um SO compartilham do mesmo *kernel* e recursos computacionais. Se houver uma falha nesse pontos, todos os contêineres ativos são afetados. As VMs estão livres dessa vulnerabilidade ?.

e) **Isolamento**: Por virtualizarem em nível de hardware, as VMs têm um isolamento superior. O isolamento realizado pelos contêiner é em nível de processo, havendo compartilhamento do *kernel*, por exemplo, como já mencionado [?, 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 contêineres. Como eles não carregam consigo um SO, e dependem do *kernel* do SO que os hospeda, não seria possível executar um contêiner com uma aplicação de certo SO se este for executado em outro SO diferente. Este não é um problema que as VMs enfrentam [?, p. 904-906].

2.2 Nuvem

A Nuvem (*Cloud*) pode ser definida como um grande conjunto de recursos virtualizados (*e.g.*, hardware, plataformas de desenvolvimento, serviços) com fácil acesso e usabilidade ?. 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 provisionamento, garantindo uso otimizado ?. Tipicamente, a utilização da nuvem é cobrada segundo um modelo denominado **pague pelo uso** (***pay-per-use***), em que os usuários pagam somente por quanto e o que usarem, segundo alguma métrica criada pelo provedor de nuvem ?. Neste modelo de cobrança, os usuários são protegidos pelo **contrato de nível de serviço** (***service level agreement***, **SLA**), 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 nuvem ?. 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 ?. Este projeto 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** (***infrastructure as a service***, **IaaS**): Nesse modelo tipicamente são oferecidos recursos de hardware (*e.g.*, processadores, memória, disco, entre outros) 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 **?**.

b) **Plataforma como Serviço** (***platform as a service***, **PaaS**): Nesse modelo, o provedor fornece 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 **?**.

c) **Software como Serviço** (***software as a service***, **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* **?**.

2.2.2 Arquitetura *Serverless*

A Arquitetura *Serverless* refere-se a um padrão de entrega de software em que aplicações, de modo geral 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 **?**: 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 aplicações. Isso é uma grande vantagem pelo fato de que, nessa arquitetura, uma aplicação pode ser escalado a zero, ou seja, se não estiver em uso, não será executada **?**. Essas características permitem ao cliente pagar somente pelo tempo que as aplicações foram de fato usadas.

Por fim, outra característica chave das Arquiteturas *Serverless* é a orientação a eventos **?**: a execução de aplicações 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 **?**.

Capítulo 3

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, as seguintes atividades:

   a) Definição de funcionalidades do sistema.

   b) Planejamento e aprimoramento da arquitetura básica do *back-end*.

   c) Estudo de ferramentas livres existentes que possam auxiliar na implantação do sistema: As atividades realizadas pelo *backend* envolverão empacotamento de código em contêineres (seguindo um paradigma *serverless*), orquestração de contêineres, escalonamento de execução de contêineres 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 seu uso.

   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