Universidade Estadual de Campinas
Instituto de Computação

Renan Monteiro Pinto Neto

Cloudboss - um sistema para orquestrar a execução de simulações computacionais na nuvem

CAMPINAS
2018

Renan Monteiro Pinto Neto

Cloudboss - um sistema para orquestrar a execução de simulações computacionais na nuvem

Dissertação apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Orientadora: Profa. Dra. Juliana Freitag Borin
Coorientador: Prof. Dr. Edson Borin

Este exemplar corresponde à versão final da Dissertação defendida por Renan Monteiro Pinto Neto e orientada pela Profa. Dra. Juliana Freitag Borin.

CAMPINAS
2018

Agência(s) de fomento e nº(s) de processo(s): CNPq, 134581/2014-8

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

P658c

Pinto Neto, Renan Monteiro, 1991-
Cloudboss - um sistema para orquestrar a execução de simulações computacionais na nuvem / Renan Monteiro Pinto Neto. – Campinas, SP : [s.n.], 2018.

Orientador: Juliana Freitag Borin.
Coorientador: Edson Borin.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Computação em nuvem. 2. Simulação computacional. 3. Microsoft Azure (Plataforma de computação). I. Borin, Juliana Freitag, 1978-. II. Borin, Edson, 1979-. III. Universidade Estadual de Campinas. Instituto de Computação. IV. Título.

Informações para Biblioteca Digital

Título em outro idioma: Cloudboss - a system for orchestrating the execution of computer simulations in the cloud
Palavras-chave em inglês:
Cloud computing
Computational simulation
Microsoft Azure (Computing platform)
Área de concentração: Ciência da Computação
Titulação: Mestre em Ciência da Computação
Banca examinadora:
Juliana Freitag Borin [Orientador]
Alexandro José Baldassin
Sandro Rigo
Data de defesa: 20-12-2018
Programa de Pós-Graduação: Ciência da Computação

Universidade Estadual de Campinas
Instituto de Computação

Renan Monteiro Pinto Neto

Cloudboss - um sistema para orquestrar a execução de simulações computacionais na nuvem

Banca Examinadora:

- Profa. Dra. Juliana Freitag Borin
  IC/UNICAMP
- Prof. Dr. Alexandro José Baldassin
  IGCE/UNESP
- Prof. Dr. Sandro Rigo
  IC/UNICAMP

A ata da defesa, assinada pelos membros da Comissão Examinadora, consta no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria do Programa da Unidade.

Campinas, 20 de dezembro de 2018

Agradecimentos

Em primeiro lugar, não posso deixar de agradecer a minha orientadora e coorientador, Professora Doutora Juliana Freitag Borin e Professor Doutor Edson Borin, por toda a paciência e empenho com que sempre me orientaram neste trabalho. Meu muito obrigado por terem me corrigido e aconselhado sempre que necessário.

Desejo igualmente agradecer a minha esposa Lais Danata Caminhoto pela compreensão e força dada desde a escolha por fazer o Mestrado até o final deste.

Gostaria também de agradecer a Unicamp e o Instituto de Computação pela infraestrutura e cordialidade fornecida. Juntamente agradecer ao CNPq pelo fomento ao trabalho realizado.

Por fim, gostaria de agradecer minha família e amigos que sempre foram exemplos para mim e sempre me motivaram para ir além.

Resumo

Simulações computacionais são fundamentais em diversas áreas para reduzir tempo, custo e riscos de testes reais, porém demandam grande poder computacional. Cada vez mais a computação em nuvem tem sido usada para satisfazer este tipo de demanda, porém, ainda há preocupações e incertezas sobre o uso, custos, implementações e segurança. Este trabalho propõe um sistema que automatiza o escalonamento e execução de simulações computacionais idempotentes em sistemas de nuvem computacional. Desse modo, diversas partes do processo são abstraídas e mesmo usuários sem muito conhecimento em infraestrutura poderão utilizar os recursos disponíveis. O sistema proposto inclui um algoritmo que auxilia na escolha da máquina de menor custo para a execução da carga de trabalho do usuário - uma das grandes dificuldades encontradas mesmo por usuários experientes. Resultados obtidos com experimentos utilizando *benchmarks* e uma carga de trabalho real apontam que o algoritmo proposto é capaz de indicar a máquina de menor custo dentre todas as configurações de máquinas disponíveis na nuvem da Microsoft, o Azure.

Abstract

Computer simulations are fundamental to reduce time, cost and risks of real tests, however, they demand great computational power. Cloud computing has been used to satisfy such demand, but there are still concerns and uncertainties about usage, costs, implementations and security. This work proposes a system that automates the scheduling and execution of idempotent computer simulation in cloud systems. In this way, several parts of the process are abstracted and even users without much knowledge in infrastructure can use the available resources. The system includes an algorithm to help the user choose the lowest-cost machine for the execution of his workload. Results from experiments using benchmarks and a real workload show that the proposed algorithm is able to indicate the lowest-cost machine considering all the available machine configurations in Microsoft's cloud provider, Azure.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

Simulações computacionais utilizam modelos matemáticos abstratos para descrever o comportamento de modelos reais em diversas áreas, tais como engenharia, extração de petróleo, aerodinâmica, entre outras. As simulações são amplamente utilizadas para prever o funcionamento de sistemas reais antes mesmo de construí-los, possibilitando a redução de custos associados com execuções de testes reais [2, 4, 5].

Para que os resultados de uma simulação sejam estatisticamente representativos, é comum executá-la milhares de vezes com diferentes entradas. Quando os modelos simulados são complexos, surge a necessidade de infraestrutura computacional de alto desempenho (HPC - do inglês - *High Performance Computing*) para conseguir executar todas essas iterações em tempo hábil [4, 5].

A crescente oferta de serviços de computação em nuvem tornou acessível sistemas com grandes quantidades de processadores. Uma nuvem computacional é um tipo de sistema distribuído e paralelo que consiste em uma coleção de computadores interconectados e virtualizados. Esses são providos dinamicamente sob demanda com base em acordos de nível de serviço (*service-level aggrements* - SLAs) estabelecidos entre consumidores e o provedor da nuvem [1]. As nuvens computacionais públicas são uma alternativa às grandes infraestruturas privadas de computação que exigem grande investimento não só na sua implantação, mas também para sua manutenção. Em nuvens computacionais o controle da infraestrutura é delegado a um provedor, sendo este responsável por sua manutenção e gerenciamento.

Dado o grande potencial - em termos de custo e desempenho - das nuvens computacionais, pesquisadores de diversas áreas têm se interessado em utilizar esses recursos, ao invés de comprar seu próprio parque computacional. Porém, há a dificuldade em utilizar, de forma eficiente, os recursos de nuvem, devido à dificuldade na configuração de ambiente e no gerenciamento de custos e à falta de conhecimento das ferramentas e serviços disponíveis para orquestrar o trabalho. É importante mencionar que a falta de conhecimento e falta de ferramentas, para ajudar na tomada de decisões, pode levar o usuário a tomar decisões erradas que elevará o custo das operações, o que pode tornar a nuvem uma opção inviável. Este trabalho foca no estudo e desenvolvimento de ferramentas que facilitem a execução de simulações em nuvens computacionais.

As simulações computacionais consideradas neste trabalho são idempotentes. O motivo para esta restrição está relacionado ao mecanismo de tolerância a falhas, que será

apresentado nos próximos capítulos. Uma tarefa é idempotente se o resultado final obtido por uma ou mais execuções desta tarefa é o mesmo. Por exemplo, considere uma tarefa de simulação que armazena o resultado em um arquivo de saída. Se cada execução da tarefa sobrescreve (ou gera, caso não exista) um arquivo com o resultado, a tarefa é considerada idempotente, pois após uma ou várias execuções o resultado será um arquivo com o resultado. Por outro lado, se cada execução da tarefa concatena o resultado obtido à um arquivo de saída, o resultado final de uma ou várias execuções não será o mesmo e a tarefa não é considerada idempotente [6].

1.1 Contribuições

Este projeto de mestrado tem como principais contribuições:

- levantamento do estado da arte em ferramentas para execução de simulações computacionais na nuvem;
- criação de um sistema auto-escalável para execução automática de simulações computacionais idempotentes na nuvem;
- desenvolvimento de algoritmo que indica a máquina com melhor custo benefício para a carga de trabalho do usuário - *Warmup*.

1.2 Organização do trabalho

Esta dissertação está organizada da seguinte maneira: o Capítulo 2 define os conceitos necessários para o entendimento deste trabalho, bem como os trabalhos relacionados. No Capítulo 3, é apresentada a infraestrutura de nuvem utilizada bem como os serviços disponíveis. O Capítulo 4 descreve o sistema proposto neste trabalho e discute detalhes de tecnologias e arquiteturas utilizadas. O Capítulo 5 apresenta diversos experimentos e resultados obtidos utilizando o sistema desenvolvido. Finalmente, o Capítulo 6 conclui a dissertação e expõe os trabalhos futuros.

Capítulo 2

Conceitos Fundamentais e Revisão Bibliográfica

Neste capítulo serão apresentados os conceitos de computação em nuvem e simulações computacionais, bem como um levantamento de trabalhos relacionados com o uso de nuvem para a execução de simulações.

2.1 Computação em nuvem

Computação em nuvem promoveu uma mudança de paradigma na computação. Neste novo paradigma a infra-estrutura de computadores está situada em um provedor e é acessada via *Internet* [7]. A principal filosofia do paradigma é entregar computação como serviço, assim como água, eletricidade, gás e telefone. Alguns autores tratam computação em nuvem como a quinta utilidade [1].

Computação em nuvem possui diversos benefícios [8] e o modelo de nuvem possui cinco características essenciais [9], sendo elas:

- **autosserviço sob demanda** - o usuário pode alocar e desalocar máquinas, serviços de armazenamento, serviços de rede, entre outros sem a necessidade de uma interação humana;
- **acesso flexível** - o acesso aos serviços pode ser feito a qualquer momento e a partir de diferentes plataformas de *hardware*. (p.ex., computadores pessoais, *smartphones*, tablets, servidores e estações de trabalho);
- **recursos compartilhados** - o provedor de nuvem deve conseguir servir diversos clientes, com diferentes requisitos de recursos físicos e virtuais, ao mesmo tempo. O usuário pode escolher onde seus recursos estarão localizados - cidade, estado e país;
- **elasticidade** - a nuvem permite que os usuários aloquem e desaloquem recursos computacionais sob demanda; desse modo, seus recursos computacionais podem ser sempre proporcionais às suas necessidades;

- **medição dos recursos** - os provedores de computação em nuvem precisam medir automaticamente o uso dos recursos que estão sendo utilizados de modo a efetuar a cobrança e fornecer métricas de uso ao cliente.

Há três modelos de serviço principais no paradigma de computação em nuvem. O primeiro é o *Infrastructure-as-a-Service* (IaaS), em que o usuário é responsável por gerenciar suas aplicações, dados, sistemas operacionais e camadas intermediárias. O segundo modelo de serviço é o *Platform-as-a-Service* (PaaS), em que a nuvem abstrai questões como camadas intermediárias e sistemas operacionais e o usuário fica responsável somente por suas aplicações e seus dados. Já o último modelo serviço é o *Software-as-a-Service* (SaaS), em que o usuário não precisa gerenciar nem os seus dados nem suas aplicações [10, 11].

As nuvens podem ser classificadas em: nuvens privadas, nuvens públicas ou nuvens híbridas. Nuvens públicas são aquelas em que a infraestrutura da nuvem é disponibilizada por algum provedor de nuvem [12, 13]. Nuvens privadas são nuvens que pertencem a uma organização sendo essa organização responsável pela gerência e manutenção da nuvem. A nuvem híbrida consiste na união de duas ou mais nuvens independentes, mas que podem trabalhar em conjunto [11, 14].

Nuvens privadas normalmente são criadas e gerenciadas através de ferramentas específicas para esse fim. Eucalyptus [15], OpenNebula [16] e OpenStack [17] são exemplos de ferramentas para gerenciamento de nuvens. Essas ferramentas realizam operações de alocação de máquinas virtuais, liberação de máquinas virtuais, gerencia de usuários, redundância, entre outras [14, 18, 19, 20].

Os consumidores de serviços de nuvem possuem diversos requisitos de qualidade de serviço diferentes. Os provedores de nuvem precisam considerar e atender esses diferentes requisitos de qualidade para satisfazer as demandas dos clientes. A Figura 2.1 mostra uma arquitetura de nuvem orientada a mercado. As principais entidades que a compõe são [1]:

- ***users* e *brokers***: são usuários da nuvem que enviam suas requisições para serem processadas;

- ***SLA resource allocator***: é responsável por diversas funções, tais como: analisar o preço dos serviços, interpretar os requisitos enviados pelos *users* e *brokers*, contabilizar a quantidade de recursos já utilizados para realizar a cobrança, monitorar quais máquinas virtuais estão disponíveis, iniciar serviços nas máquinas virtuais e monitorar a execução das requisições;

- ***virtual machines (VMs)***: várias máquinas virtuais podem ser inicializadas e finalizadas sob demanda devendo haver flexibilidade para compartilhar recursos de uma mesma máquina física entre várias máquinas virtuais, sempre respeitando os requisitos de qualidade do cliente;

- ***physical machines***: máquinas físicas para atender a demanda dos clientes.

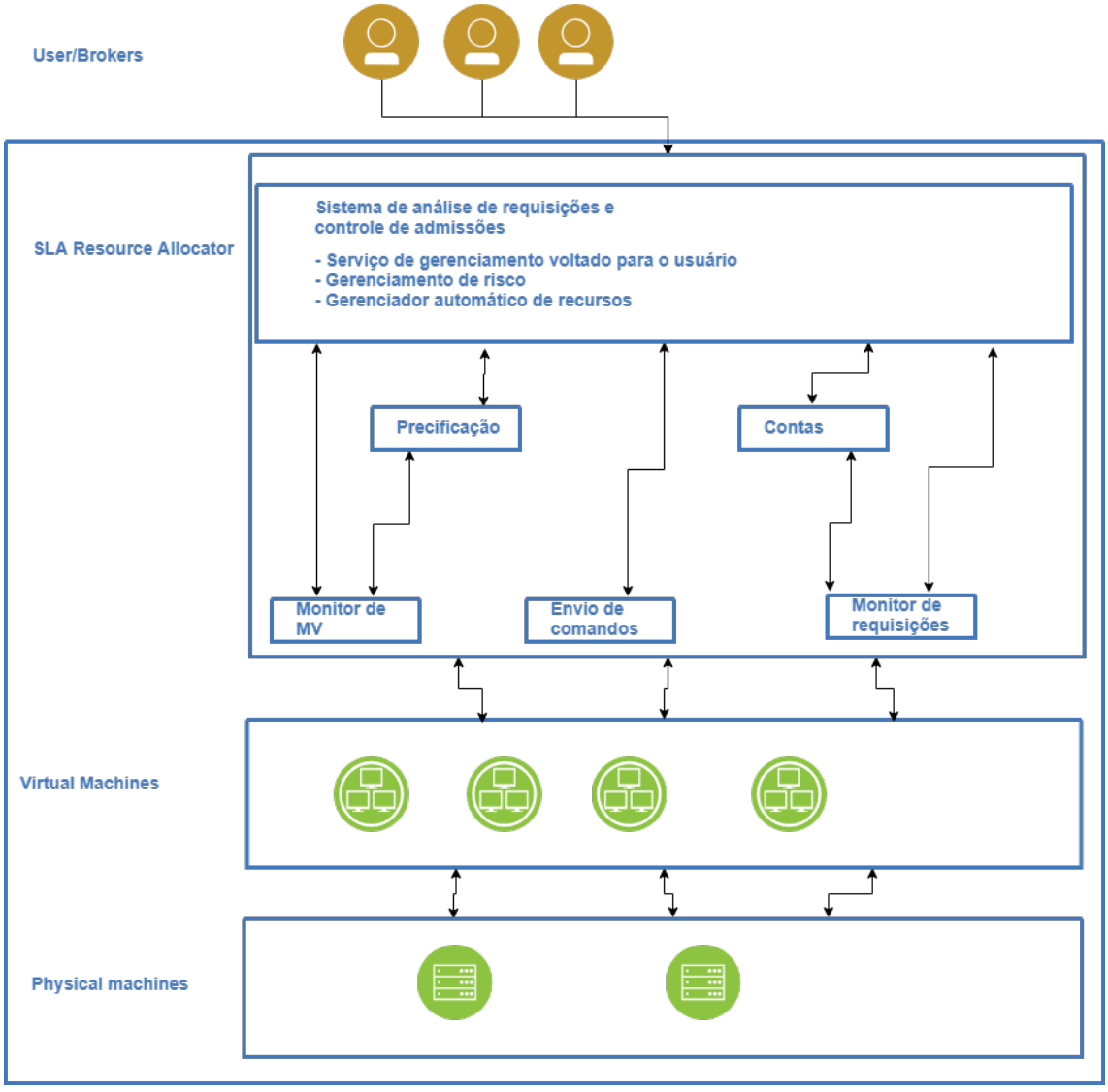

Figura 2.1: Arquitetura em alto nível de uma nuvem orientada a mercado [1].
Fonte: Adaptado de Buyya *et al.* [1].

2.2 Simulações computacionais

Uma simulação consiste em, basicamente, três etapas, como pode ser visto na Figura 2.2. O *Modeling stage*, ou etapa de modelagem, consiste em transformar um modelo matemático em um modelo para simulação por meio de ferramentas de modelagem. O *Execution stage*, ou etapa de execução, utiliza o modelo gerado na etapa de modelagem e o cenário para essa simulação como entradas para as ferramentas que executarão essa simulação, tendo como saída o resultado dessa simulação. O último estágio, o *Analysis stage*, ou etapa de análise, interpreta o resultado da etapa anterior para mostrar ao usuário os dados que este quer saber de uma maneira mais simples e visual. Nesse estágio também é possível editar e/ou criar novos cenários que serão utilizados em uma nova etapa de execução [2].

A etapa de execução costuma ser uma etapa extremamente custosa computacionalmente, pois deve executar milhares de vezes o modelo criado para várias possíveis entradas para o problema. Assim, será possível verificar como o modelo se comporta dado diversas situações que podem ocorrer. Há diversos modelos que são idempotentes e independentes logo, a ordem e quantidade de vezes que suas entradas são executados não são importantes, pois sempre geram os mesmos resultados.

2.3 Trabalhos relacionados

Liu *et al* [2] propõem uma ferramenta chamada CSim - *Cloud-based computer Simulation* - para realização de simulações na nuvem. A arquitetura do CSim é dividida em: usuários, *web browser* e a nuvem de simulação. Os usuários enviam seus modelos e simulações por meio do *web browser* para a nuvem de simulação. A nuvem de simulação possui diversos componentes que oferecem suporte para tarefas de modelagem como serviço, execução como serviço e análise como serviço. O SIMaaS (SIMulation as a Service) fornece aos usuários serviços como modelagem, execução e análise das simulações. O Módulo de Gerenciamento da Nuvem (CMM) inclui módulos para gerenciamento de páginas *Web*, usuários, segurança e alocação de recursos. O IaaS fornece camadas e serviços de virtualização das máquinas físicas. A Figura 2.3 ilustra a arquitetura proposta pelos autores.

Jorissen, Vila e Rehr [21] criaram uma interface que facilita operações básicas na nuvem computacional da Amazon [12] tais como: conectar-se ao serviço, criar novas instâncias de máquinas, eliminar instâncias, acessar arquivos, iniciar tarefas, entre outras. Os provedores de nuvem definem instâncias como as máquinas virtuais que estão sendo fornecidas. Os autores criam também uma interface gráfica para realizar simulações de materiais e química quântica, bem como uma interface para configuração da nuvem onde é possível informar usuário, senha e arquivos de configuração. Por fim, os autores mostram que o processo de iniciar/configurar as instâncias é paralelizável e com isso é possível ter um ganho de tempo de aproximadamente duas vezes na etapa da inicialização das instâncias, em comparação com a versão serial.

Angeline e Masala [22] propõem a criação de uma ferramenta para aceleração de simulações de comunicações multimídia utilizando computação em nuvem. Os autores

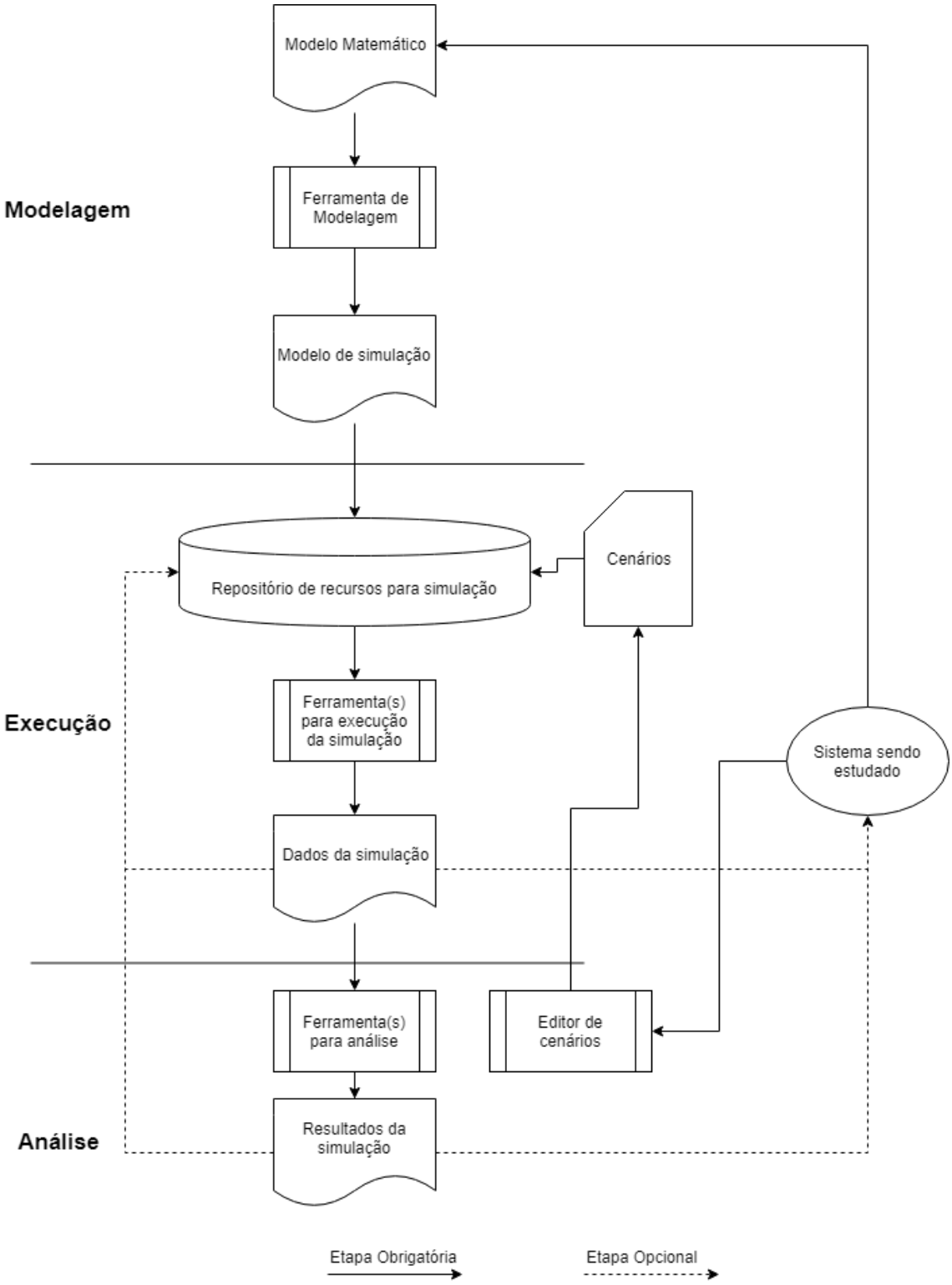

Figura 2.2: *Workflow* de uma simulação [2].
Fonte: Adaptação de Liu *et al.* [2].

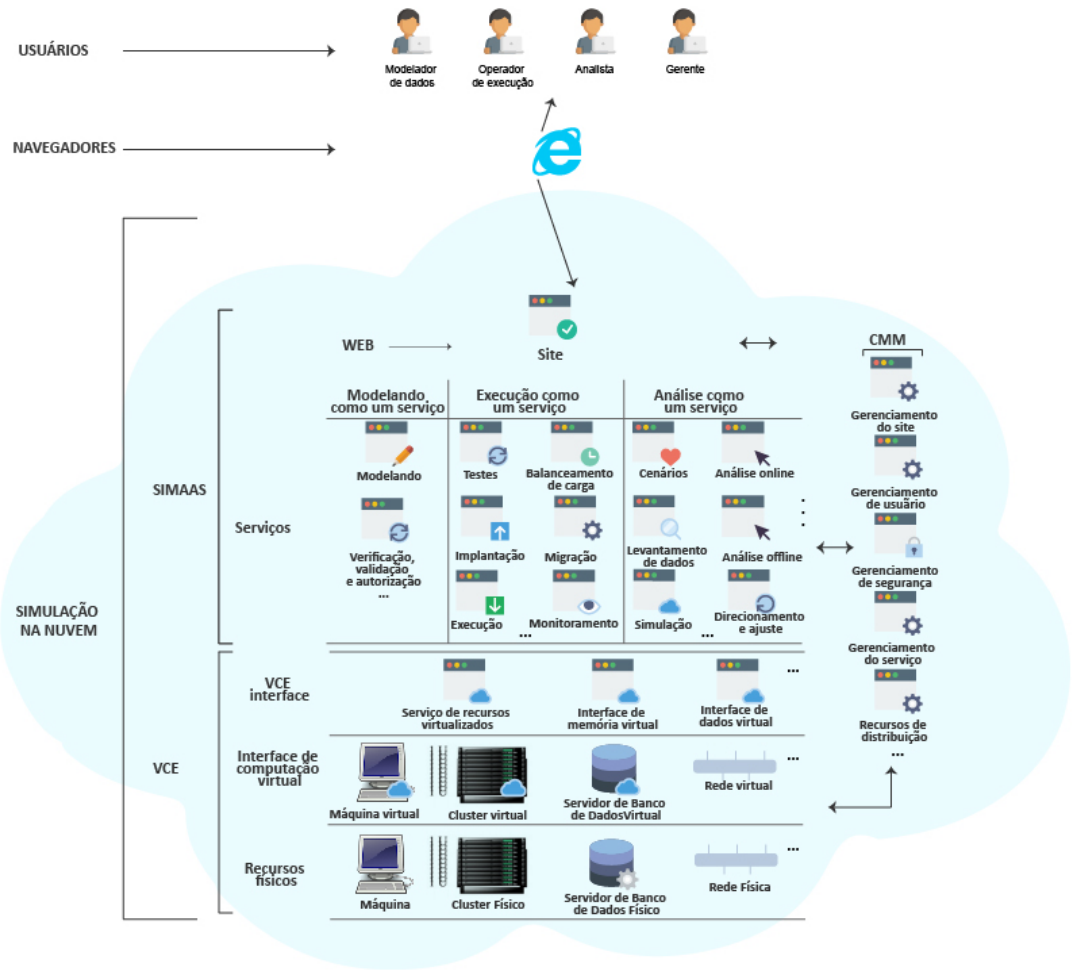

Figura 2.3: Arquitetura do CSim [2].
Fonte: Adaptação de Liu *et al.* [2]

também citam a importância da paralelização na etapa de inicialização e configuração das instâncias. Porém, neste caso, os autores apontam que quanto mais tempo uma instância fica esperando por outra, mais caro será o serviço, visto que na nuvem computacional se paga pelo tempo de alocação de cada máquina, independente se ela está realizando ou não computações. Outro ponto importante levantado pelos autores é que deve-se avaliar o custo do uso de nuvem computacional em comparação com a compra de servidores próprios. Para isso deve-se levar em consideração diversas variáveis, tais como: desgaste e deterioração das máquinas, atualizações de *hardware*, resfriamento, entre outras. Os autores também destacam a importância de se utilizar instâncias especializadas para o problema em questão. Para problemas que possuam a memória como fator limitante para o desempenho, por exemplo, deve-se utilizar instâncias especializadas em desempenho de memória.

No trabalho feito por Jakovits, Srirama e Kromonov [23] foi proposta uma ferramenta para simulações científicas utilizando nuvem computacional. Essa ferramenta é baseada no modelo BSP - *Bulk Synchronous Parallel* - que é um é um modelo de programação para algoritmos paralelos iterativos. Esse modelo é dividido em super passos e cada super passo é dividido em: computação local, sincronização e barreira global. Nesse trabalho os autores se preocuparam com falhas e tentaram otimizar o modelo, visto que em certos pontos da execução algumas tarefas podem ficar esperando outras tarefas menores. O grande problema dessa ferramenta é que utiliza a biblioteca legada Oxford BSPlib [24]. A última atualização desta biblioteca foi em 1998 e apresenta problemas na instalação em computadores modernos, principalmente em sistemas de 64 *bits* [23].

Sadooghi *et al.* [25] realizaram testes para verificar a viabilidade de executar aplicações científicas em nuvem. Os autores utilizam a nuvem da AWS [12] e a FermiCloud. FermiCloud é uma *IaaS* pertencente ao Fermilab, um laboratório americano de física de partículas [26] especializado em computação de alto desempenho. Para os experimentos os autores executaram diversos *benchmarks* para verificar o desempenho de máquinas de diferentes configurações. Com os resultados obtidos os autores concluem que aplicações que dependem muito de interações entre máquinas não possuem um bom desempenho na nuvem, chegando a 70% de desempenho se comparado a execução de uma única máquina física. Em contrapartida, os autores concluem que aplicações que não dependem de resultados externos, possuem o desempenho dentro do esperado, baseado na configuração fornecida pelo provedor se comparado a uma máquina física com a mesma configuração.

Yang *et al.* [27] apontam a importância de um sistema de auto escalabilidade para reduzir custos e manter o SLA. Com essa motivação, os autores propõem um modelo de regressão linear para prever a carga de trabalho a que as máquinas virtuais serão submetidas. Este modelo utiliza dados de cargas de trabalho anteriores para prever qual será a carga de trabalho de uma próxima iteração. O sistema é dividido em três partes que, normalmente, se repetem em um ciclo de cinco minutos:

1. Verifica se a quantidade de máquinas virtuais em uso é suficiente para atender a atual demanda. Caso não seja, aumenta as configurações das máquinas atuais para suprir a demanda.

2. Faz a previsão da próxima iteração. Caso o número atual de máquinas não consiga

Tabela 2.1: Comparação entre os trabalhos relacionados e este trabalho.

Autores	Tarefas dependentes	Tarefas idempotentes	Tolerância a falhas	Auto escalonamento	Otimização de custos
Jorissen, Vila e Rehr	•				
Angeline e Masala	•				•
Jakovits, Srirama e Kromonov	•		•		•
Sadooghi *et al.*	•	•			•
Yang *et al.*				•	•
Jonas *et al.*		•	•	•	•
Este trabalho		•	•	•	•

atender a demanda prevista, dispara a criação de novas máquinas.

3. Caso a previsão seja de um número menor de máquinas, o sistema desliga máquinas para evitar desperdício de recursos e financeiro. Ao término desta operação, o ciclo se repete.

Segundo os experimentos executados, essa abordagem prediz melhor a carga de trabalho, custa menos e causa menos impacto no SLA, comparado a outros escalonadores similares. Apesar dos resultados melhores, os autores não explicitam quão melhor foi o resultado.

O trabalho realizado com Jonas *et al* [28] tem como objetivo facilitar e auxiliar na execução de tarefas idempotentes. Para tal, foi desenvolvido um protótipo chamado PyWren [29]. Esse protótipo tem a capacidade de executar tarefas utilizando o sistema Lambda [1] e S3 [2] da Amazon Web Service. Assim, é possível executar até 3000 tarefas de forma simultânea. Os autores também resolvem o problema de tolerância a falhas, simplesmente reexecutando a tarefa que falhou. Os pontos negativos desta abordagem são:

1. limite de 3000 tarefas;

2. funções e aplicações que exigem recursos complexos não são compatíveis, devido a problemas em instalações de bibliotecas entre outros;

3. execução da tarefa não pode ultrapassar 300 segundos, isto inclui o tempo para adquirir os arquivos de entrada, enviar os de saída e realizar qualquer configuração.

Os trabalhos citados resolvem problemas complexos. Alguns ajudam ao usuário entender melhor como sua nuvem irá funcionar. Outros desenvolvem aplicações que auxiliam a execução de tarefas de forma sincronizada. Além de desenvolver ferramentas, alguns

[1] Serviço que permite executar uma determinada tarefa sem precisar fornecer máquinas ou outros tipos de gerenciamento

[2] Serviço de armazenagem de dados

executam *benchmarks* para relacionar desempenho com custo. Esses dados estão sumarizados na Tabela 2.1. Nela é possível notar a preocupação em otimizar os custos ao se utilizar os recursos da nuvem, cuja importância é discutida pelos autores Angeline e Masala [22]. Outro ponto importante é a utilização de tarefas que trocam, ou esperam, informações entre si.

Este trabalho se preocupa em otimizar os custos operacionais em nuvem usando uma abordagem semelhante ao trabalho de Sadooghi *et al* [25]. A principal diferença é que os testes são executados para a carga de trabalho do usuário, gerando um resultado personalizado e mais preciso que será discutido no Capítulo 4. Outra preocupação é realizar o auto escalonamento das máquinas para evitar desperdício de recursos e custos, além da orquestração da carga de trabalho. Este trabalho não utiliza o sistema Lambda - da Amazon Web Services - para executar suas operações, portanto não possui as mesmas limitações que o trabalho desenvolvido por Jonas *et al* [28].

Capítulo 3

Infraestrutura

Este capítulo apresenta os componentes utilizados na infraestrutura do sistema e discute os pontos positivos e as limitações de cada componente.

3.1 Microsoft Azure

O serviço de nuvem pública da Microsoft, o Microsoft Azure [13], é um serviço flexível e eficiente. Com ele é possível consumir diversos serviços de computação que estão em todas as camadas do desenvolvimento de uma aplicação [30]. Também é possível criar, de forma prática, serviços PaaS e IaaS, utilizando *Command Line Interface (CLI)*[1], requisições REST ou um sistema de navegação *Web* poderoso, simples e intuitivo.

Por esses meios é possível realizar diversas configurações, tais como: auto escalabilidade, opções de desenvolvimento, configurações de serviços, ajustes específicos para cada serviço e outros [31]. Desse modo, tanto usuários inexperientes quanto usuários avançados tem possibilidade de configurar e aproveitar os serviços básicos contratados de acordo com suas necessidades.

Para este projeto de mestrado, diversos componentes da Microsoft Azure foram utilizados. O intuito foi fazer uso de serviços disponíveis que abstraem diversas tarefas, delegando à plataforma responsabilidades que levariam muito tempo para serem desenvolvidas e que demandariam alto custo de manutenção. As subseções seguintes descrevem esses componentes.

3.1.1 Cloud Service

Cloud Service é um serviço que oferece funcionalidades e abstrações para outros sub-serviços. Entre estes sub-serviços está a camada responsável por gerenciar as máquinas virtuais. Tem grande importância para a aplicação, pois possibilita o acesso remoto às máquinas virtuais disponíveis e aos dados para monitoramento das máquinas.

Outra funcionalidade importante é o auto escalonamento, responsável pela elasticidade das máquinas virtuais. Regras de elasticidade podem ser criadas pelo usuário para definir quantas máquinas serão ligadas ou desligadas, com base em políticas personalizadas, ou

[1]Nome dado a interações feitas com um programa via múltiplas linhas de comando executadas sucessivamente

em métricas disponíveis, tais como uso de processador e quantidade de tarefas em uma fila. A Figura 3.1 apresenta um exemplo de configuração de uma regra de elasticidade. Nesta figura a regra está baseada em uso da CPU. Foi definido, no campo *INSTANCE RANGE*, que o número mínimo de máquinas deve ser 3 e o máximo 10. Além do número de máquinas, também é configurado quando se deve ligar ou desligar máquinas; para o caso deste exemplo, máquinas serão desligadas caso a média do uso de CPU seja menor do que 60% e máquinas serão criadas caso o uso da CPU esteja maior do que 80%, como visto no campo *TARGET CPU*. Os campos *SCALE UP BY* e *SCALE UP WAIT TIME* determinam, respectivamente que podem ser criadas 2 máquinas por vez e que o intervalo entre a criação de novas máquinas deve ser de 20 minutos. A mesma lógica se aplica para o desligamento de máquinas, aqui definidas pelos campos *SCALE DOWN BY* e *SCALE DOWN WAIT TIME*.

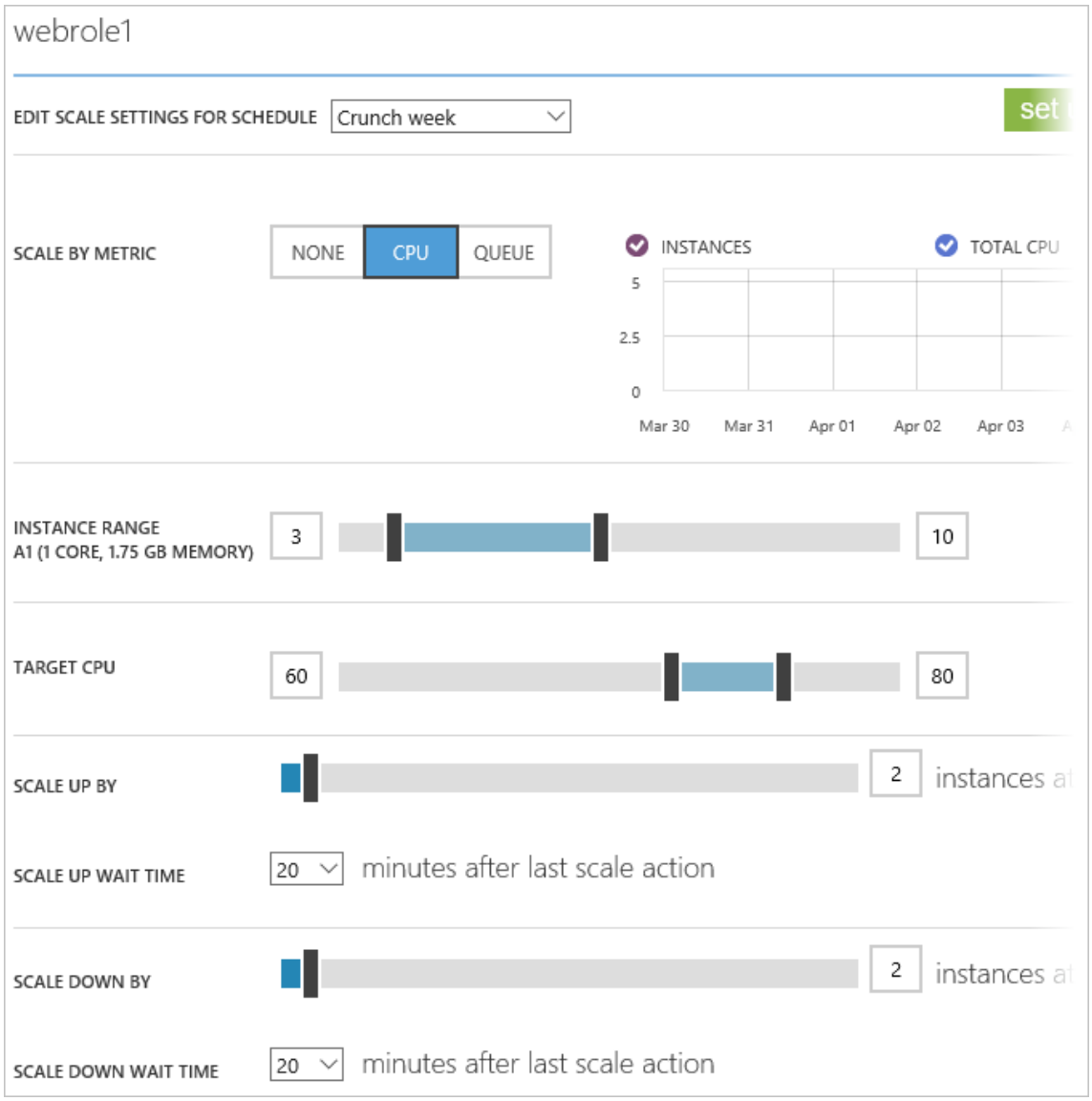

Figura 3.1: Tela de configuração de escalonamento.
Fonte: Captura de tela de configuração de escalonamento do Microsoft Azure [13].

Como limitações deste serviço vale ressaltar que o valor mínimo de máquinas é um,

ou seja, sempre haverá pelo menos uma máquina ligada. Outro ponto a se observar é que não é possível mudar a configuração (CPU, RAM e etc) das máquinas para aquele grupo auto escalável. Além disso, cada Cloud Service possui limite de 50 (cinquenta) máquinas virtuais, portanto caso seja preciso usar mais de 50 máquinas, é necessário criar mais de um Cloud Service.

3.1.2 Máquinas Virtuais

Máquinas virtuais estão cada dia mais presentes em servidores. Com elas é possível prover um ambiente capaz de abstrair o hardware para o sistema operacional sendo ainda possível que várias máquinas virtuais operem sobre o mesmo hardware [32]. Desse modo, provedores de serviços em nuvem podem alocar várias máquinas virtuais com diferentes configurações em um mesmo hardware. Máquinas virtuais de diferentes usuários também podem compartilhar uma mesma máquina física. Neste caso, o provedor deve garantir o isolamento e a privacidade entre as máquinas virtuais.

É possível contratar máquinas virtuais com diversas configurações de *hardware* para atender às necessidades de diversas aplicações e usuários. Essas máquinas são divididas em diversas classes, tais como: computação de propósito geral, especializadas em computação, focadas em computação de alto desempenho, com placas de vídeo e com rede otimizada. O preço e a disponibilidade variam de acordo com a localização física e a configuração da máquina. A Tabela 3.1 traz as principais informações técnicas sobre as máquinas.

Essas máquinas possuem propósitos diferente uma das outras. As máquinas de A0 até A7 são dedicadas para computação de propósito geral. Máquinas A8 até A11 possuem foco em processamento, porém as A8 e A9 possuem uma rede mais poderosa. As máquinas da família D também são máquinas de propósito geral, porém possuem armazenamento com SSD e seus processadores são um pouco melhores. Ainda na família D, temos máquinas com o sufixo V2, que indica uma segunda geração da família D com processadores mais recentes. Por fim, temos a família G que são máquinas com processadores poderosos e grande quantidades de recursos de armazenamento e memória. É importante notar que conforme os recursos e poder de processamento aumentam, seu custo também aumenta.

3.1.3 Captura de Imagens de Máquinas Virtuais

Com este serviço é possível capturar o estado atual da máquina virtual. Isto inclui toda a pilha de *software* e dados que estiverem na máquina, quando a captura for realizada.

Para realizar a captura, basta selecionar a máquina virtual desejada, escolher a opção em capturar, definir os nomes e confirmar, como mostrado nas Figuras 3.2 e 3.3. Na sequência, a imagem desta máquina já estará disponível para uso, como aparece na Figura 3.4. Dessa forma, toda vez que uma máquina virtual for criada a partir desta imagem, a nova máquina será uma cópia exata da máquina copiada no instante da operação.

3.1.4 Queue Service

O *Queue Service*, ou serviço de fila, é responsável por armazenar mensagens. Estas mensagens possuem tamanho máximo de 64KB e podem ser acessadas de qualquer lugar

Tabela 3.1: Máquinas virtuais disponíveis para uso no Azure.

Máquina	Núcleos	Memória Ram (GB)	Disco (GB)	Preço (Dólar/hora)
A0	1 (compartilhado)	0,75	20	0,02
A1	1	1,75	70	0,006
A2	2	3,5	135	0,12
A3	4	7	285	0,24
A4	8	14	605	0,48
A5	2	14	135	0,25
A6	4	28	285	0,5
A7	8	56	605	1
A8	8	56	382	0,975
A9	16	112	382	1,95
A10	8	56	382	0,78
A11	16	112	382	1,56
D1	1	3,50	50	0,077
D2	2	7	100	0,154
D3	4	14	200	0,308
D4	8	28	400	0,616
D11	2	14	200	0,195
D12	4	28	200	0,39
D13	8	56	400	0,78
D14	16	112	800	1,542
D1V2	1	3,5	50	0,085
D2V2	2	7	100	0,171
D3V2	4	14	200	0,342
D4V2	8	28	400	0,684
D5V2	16	56	800	1,368
D11V2	2	14	100	0,224
D12V2	4	28	200	0,449
D13V2	8	56	400	0,897
D14V2	16	112	800	1,773
G1	2	28	384	0,61
G2	4	56	768	1,22
G3	8	112	1536	2,44
G4	16	224	3072	4,88
G5	32	448	6144	8,69

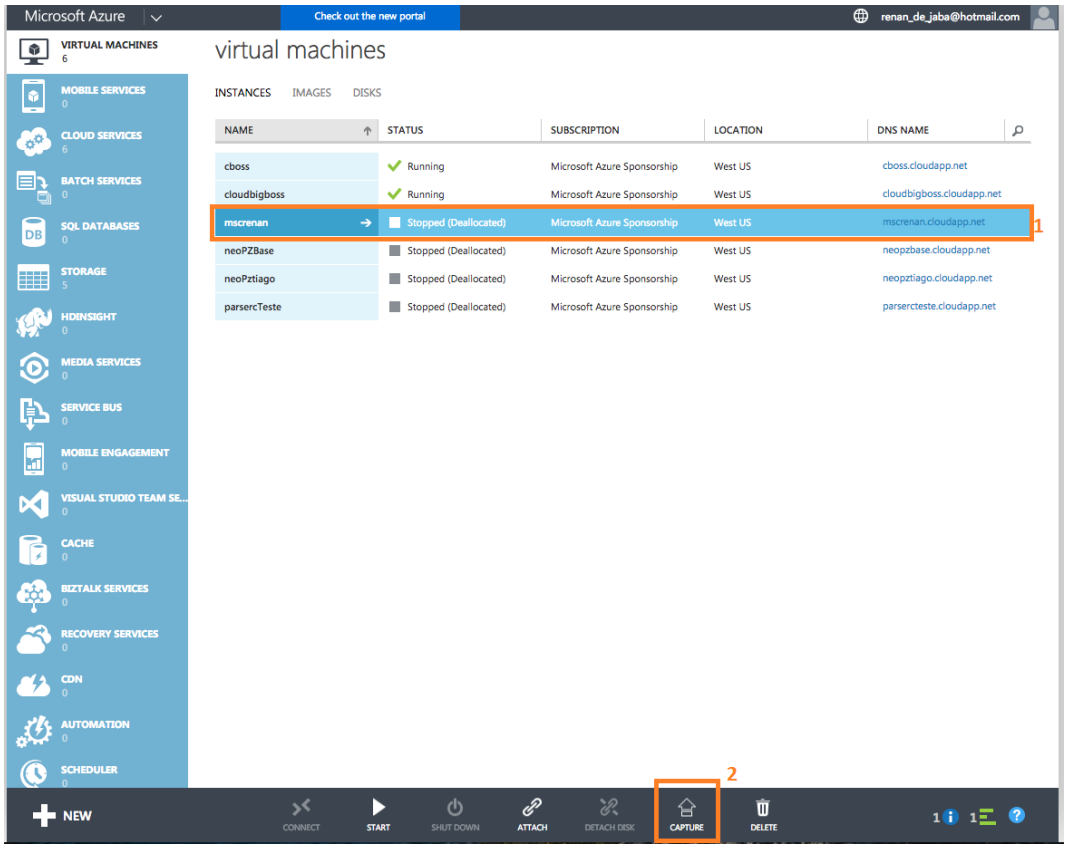

Figura 3.2: (1) Escolha da máquina. (2) Opção para capturar imagem da máquina virtual.
Fonte: Captura de tela na área de seleção e opções na para máquinas virtuais.

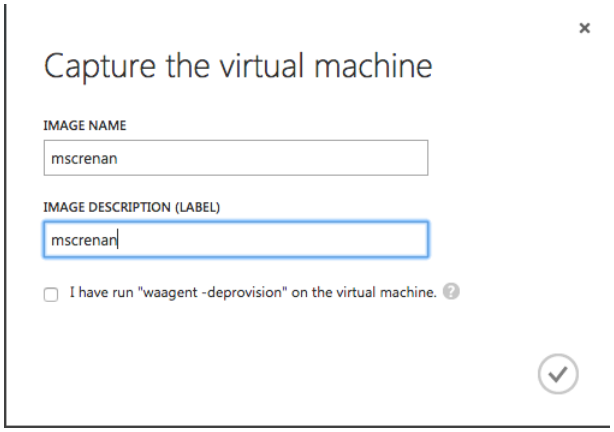

Figura 3.3: Escolha de nomes e descrições para a imagem que será criada.
Fonte: Captura de tela para captura de imagem de máquina virtual.

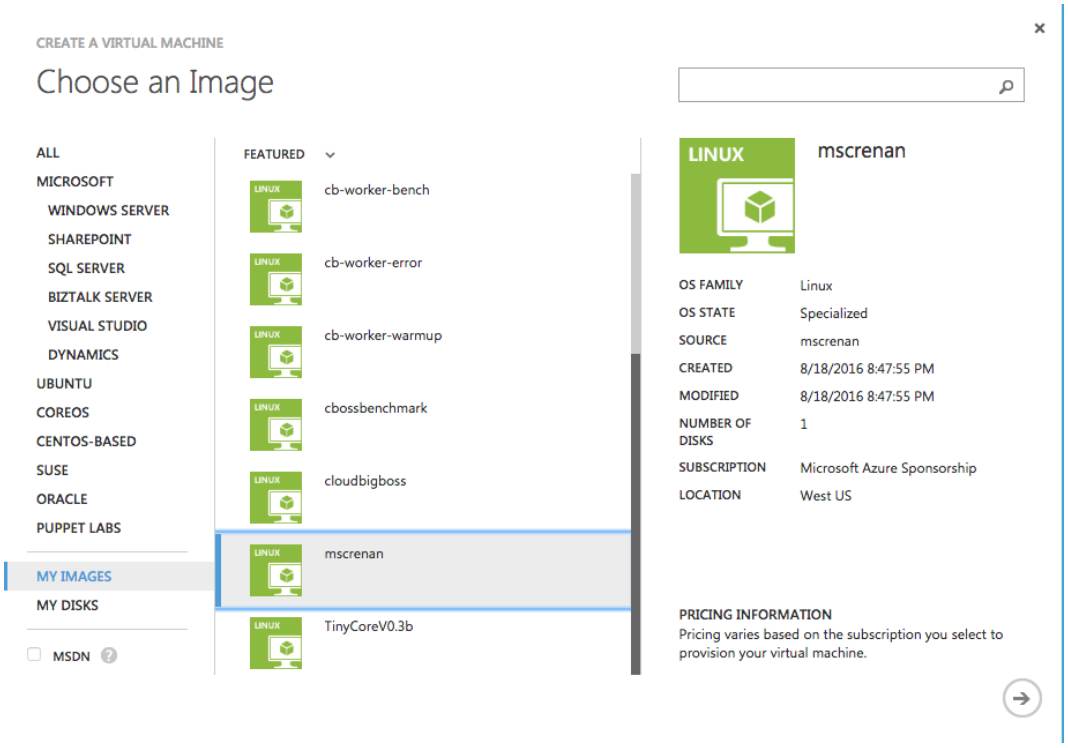

Figura 3.4: Escolhendo imagem para a máquina virtual personalizada.
Fonte: Captura de tela para escolha de imagem para criar máquina virtual.

do mundo. O serviço possui capacidade bastante elevada, podendo suportar 5TB de dados e uma taxa de 2000 mensagens por segundo (para mensagens de 1KB).

Este serviço é normalmente usado por aplicações com arquiteturas voltadas a alta disponibilidade. Assim, toda requisição que chega à aplicação vai para uma fila, o que evita que requisições sejam perdidas e cria um *pool* de tarefas para serem consumidas.

Outra dinâmica no serviço de fila é o processo de consumo. Quando uma mensagem é solicitada, o solicitante recebe a mensagem e esta se torna invisível na fila. Para efetivar o consumo, o solicitante precisa confirmar o seu recebimento e só então a mensagem deixa de existir na fila. Caso a confirmação não seja recebida, a mensagem volta a ficar ativa na fila após cinco minutos, que é o tempo padrão e que também pode ser ajustado.

3.1.5 Blob Storage

O serviço de *Blob Storage* tem como objetivo armazenar qualquer tipo de dado, seja ele estruturado ou não. Estes arquivos podem ser acessados de qualquer lugar do mundo via HTTP e HTTPS. Além disso, estes dados podem estar disponíveis publicamente ou apenas para usuários com acesso liberado. Isto permite que sistemas disponibilizem arquivos estáticos para seus usuários por meio do navegador, como bibliotecas JavaScript, imagens e vídeos.

Os *Blobs* são armazenados seguindo a estrutura da Figura 3.5. A **Conta** representa uma conta de armazenamento, podendo inclusive ser a mesma da *Queue* (Seção 3.1.4). Dentro de uma conta você pode ter vários contêiners. Estes contêiners funcionam como pastas em um sistema comum de arquivos, com a restrição de que não há contêiners dentro

de outros contêiners. Dentro de um contêiner o usuário pode armazenar um número ilimitado de *blobs* (arquivos).

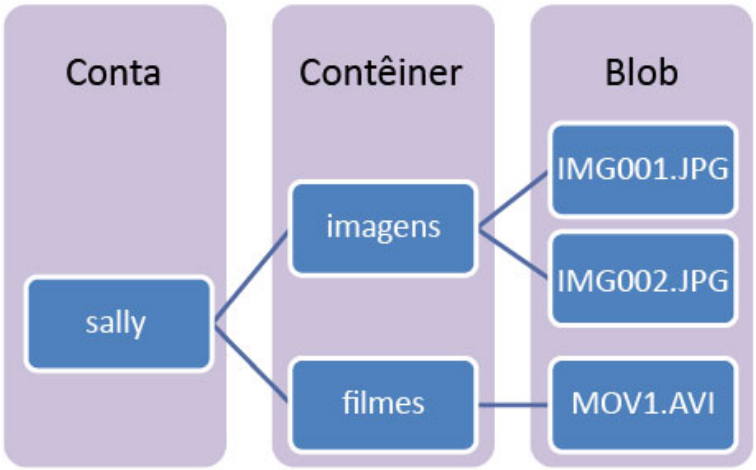

Figura 3.5: Arquitetura de armazenamento dos arquivos no *Blob Storage*.
Fonte: Microsoft Azure

3.2 Microsoft Azure *vs* Amazon Web Services (AWS)

Assim como o Microsoft Azure, o Amazon Web Services [12] é um provedor de nuvem pública oferecido pela Amazon. Ambas possuem e fornecem serviços semelhantes em diversas áreas, normalmente mudando apenas preço, limitações e algumas configurações. Assim sendo, todos os serviços utilizados por este trabalho poderiam ser trocados por serviços fornecidos pela AWS, guardado as devidas adaptações. As substituições seriam:

- **Máquinas Virtuais e Cloud Service:** A Amazon oferece o Elastic Compute Cloud (Amazon EC2). Com ele é possível criar máquinas virtuais e ter acesso a um painel de controle. Neste painel, assim como no Azure, é possível configurar o sistema de auto-escalabilidade além de outros ajustes.

- **Queue Service:** O Amazon Simple Queue Service (SQS) é um serviço de filas de mensagens simples e escalável. Sua grande diferença para o *Queue Service* é o tamanho limite de suas mensagens. Sendo o limite de 256KB, ao invés de 64KB oferecido pelo Microsoft Azure.

- **Blob Storage:** Pelo Amazon Simple Storage Service (Amazon S3) é possível armazenar arquivos. Dentro de cada bucket - análogo a um contêiner - o usuário pode armazenar um número ilimitado de arquivos, que podem estar disponíveis para um público geral ou somente para usuários com autorização. Sendo assim, os arquivos ficam disponíveis para serem consumidos de qualquer lugar do mundo.

É possível estabelecer relações entre os serviços disponibilizados pela Amazon e pela Microsoft. A escolha entre as plataformas normalmente se dá por custos, facilidades no uso e motivos pessoais. Logo, este trabalho de mestrado pode ser adaptado, ou estendido, para utilizar os recursos da Amazon.

3.3 Node.js

Javascript é uma linguagem de programação primeiramente desenvolvida para executar em navegadores, com a função de criar uma interação entre a página *Web* e o cliente. É uma linguagem de programação interpretada e com tipagem fraca. Após sua popularização, foi desenvolvido um programa para interpretar e executar o Javascript do lado do servidor, chamado de Node.js. Foi desenvolvido para ser escalável e dirigido a eventos, ou seja, responde bem em ambientes que exigem respostas rápidas, quando algum evento precisa ser tratado, como requisições *Web*.

Basicamente todo o sistema construído para este trabalho de mestrado foi desenvolvido utilizando Node.js. Foram usadas diversas bibliotecas, entre elas ferramentas proprietárias da Microsoft e ferramentas para facilitar processos, abstraindo operações.

3.4 *Representational State Transfer*

REST - *Representational State Transfer*, ou em português Transferência de Estado Representacional - é uma arquitetura para comunicação entre aplicações utilizando a *Internet* que utiliza o protocolo HTTP, ou HTTPS, para enviar ou receber dados. Seu uso é muito difundido em API's (*Application Programming Interface*) que consomem serviços via *Internet*. Desde que os *endpoints* utilizem REST, não importa qual tecnologia ou linguagem de programação está recebendo ou enviando dados. Como exemplo pode-se utlizar o serviço de fila do Azure. Todas as mensagens são consumidas e enviadas por meio de um *Endpoint*, seja um programa, ou um *framework*, independente da linguagem de programação.

Neste projeto REST foi utilizado para a comunicação com as máquinas e os serviços do Microsoft Azure. Foi utilizado um *framework* disponibilizado pela Microsoft, para lidar com operações de envio e recebimento de arquivos e mensagens. Para as demais operações, foram utilizadas chamadas diretas para os servidores da Azure, como funções para desligar e ligar máquinas, criar e deletar *Cloud Services*, entre outros.

3.5 Considerações

Os serviços utilizados neste trabalho são estáveis, confiáveis e escaláveis. Logo, foi possível abstrair e delegar funcionalidades, que precisariam ser implementadas, ao provedor de nuvem. Isso acelerou o processo de desenvolvimento e garantiu uma maior confiabilidade do sistema e seus recursos.

Capítulo 4

CloudBoss

Este capítulo explica os detalhes tecnológicos, objetivos, limitações e funcionamento do sistema desenvolvido.

4.1 Foco do sistema

O sistema desenvolvido neste trabalho de mestrado possui um foco bem claro: executar programas independentes e idempotentes, de maneira distribuída em nuvem - neste caso, na Microsoft Azure (Seção 3.1). Um programa é considerado idempotente se, e somente se, o resultado gerado, para uma determinada entrada, é sempre o mesmo independente do número de vezes que ele é executado [33]. Logo, uma falha não impactará no resultado final da carga de trabalho, no caso em que tarefas sejam executadas mais de uma vez. Caso um conjunto de tarefas não seja independente, seria necessário controlar a ordem em que são executas, além de se preocupar com os dados que são trocados entre as tarefas.

É foco também, fazer com que os passos para a execução deste programa sejam simples e intuitivos, mesmo para pessoas não familiarizadas com ferramentas de nuvem. Assim sendo, é necessário que o sistema abstraia serviços e configurações. Vale lembrar que ainda assim é necessário algumas intervenções e configurações por parte dos usuários, como por exemplo configurar a máquina virtual que o seu programa irá executar (configurar variáveis de ambientes, bibliotecas e outros).

Além da execução de tarefas, o sistema também possui o objetivo de escolher a máquina ideal. Por meio de metodologia proposta neste trabalho, que será explicada na Seção 4.3.2, o sistema faz uma sugestão de qual máquina é a melhor para executar a carga de trabalho. A definição aqui usada como melhor máquina, não é necessariamente a máquina em que a aplicação executa mais rápido, mas sim aquela que possui melhor relação custo-benefício.

4.2 Arquitetura do sistema

A arquitetura do sistema foi desenhada para trabalhar de forma sincronizada e distribuída. Portanto, cada componente possui uma função bem definida e diferentes componentes não precisam operar necessariamente em uma mesma máquina, ou mesmo em um mesmo

provedor de nuvem. Algumas modificações precisam ser feitas, pois atualmente o único provedor, ou máquina, aceita é pelo Microsoft Azure, porém, a máquina mestra, que será descrita mais adiante, pode ser qualquer máquina, inclusive uma máquina pessoal.

O primeiro componente da arquitetura é a máquina que gerencia os trabalhos, ou máquina mestra. Ela pode ser vista na Figura 4.1. Como já mencionado, esta máquina não precisa estar necessariamente dentro do Azure, mas é importante ressaltar que o tempo de envio de tarefas para a fila e para o Storage está diretamente ligado à qualidade de conexão entre a máquina e a nuvem. Portanto, para cargas de trabalho que envolvem arquivos grandes, é recomendado que esta máquina fique no mesmo provedor. Este componente da arquitetura é responsável por:

- prover uma interface gráfica para o usuário, que pode ser acessada por qualquer navegador, inclusive por aparelhos celulares;
- criar tarefas e enviá-las para a fila de tarefas;
- em casos em que a tarefa possui algum arquivo de entrada, enviar o arquivo de entrada para o módulo de armazenamento - este passo é realizado antes do envio da tarefa para a fila;
- criar novas máquinas, excluir máquinas, desligar e religar máquinas;
- gerenciar a escalabilidade das máquinas - por meio de uma política de escalabilidade, ele determina quantas máquinas precisam estar ligadas;
- identificar falhas na execução de tarefas e decidir como tratá-las - por exemplo executá-las em uma máquina com maior poder computacional, ou desconsiderá-las.

O segundo componente da arquitetura é responsável pelo armazenamento de informações. Para fins de explicação, vamos separar em três sub serviços, que possuem funções diferentes e independentes. Na parte superior da Figura 4.1, vemos o banco de dados, responsável por armazenar os arquivos de entrada, quando necessário, e os arquivos de saída. Quando uma tarefa é criada, ela pode possuir um arquivo de entrada. Este arquivo é carregado para o banco de dados. Na fase de execução da tarefa, se há um arquivo de entrada, este é copiado para a máquina responsável pela execução. Ao final da execução é esperado que a tarefa gere um arquivo de saída. Este arquivo de saída é enviado novamente para o Banco de dados. Desse modo, o usuário pode coletar os dados gerados pela carga de trabalho.

O segundo sub serviço pode ser encontrado na parte inferior da Figura 4.1. O Armazenamento de Filas é responsável por armazenar mensagens, que neste projeto representam tarefas. Na etapa de criação da tarefa, o sistema gera uma mensagem que contém diversas informações, entre elas:

- identificador da mensagem;
- data do envio;
- um sinalizador que demonstra se aquela tarefa possui ou não arquivo de entrada;

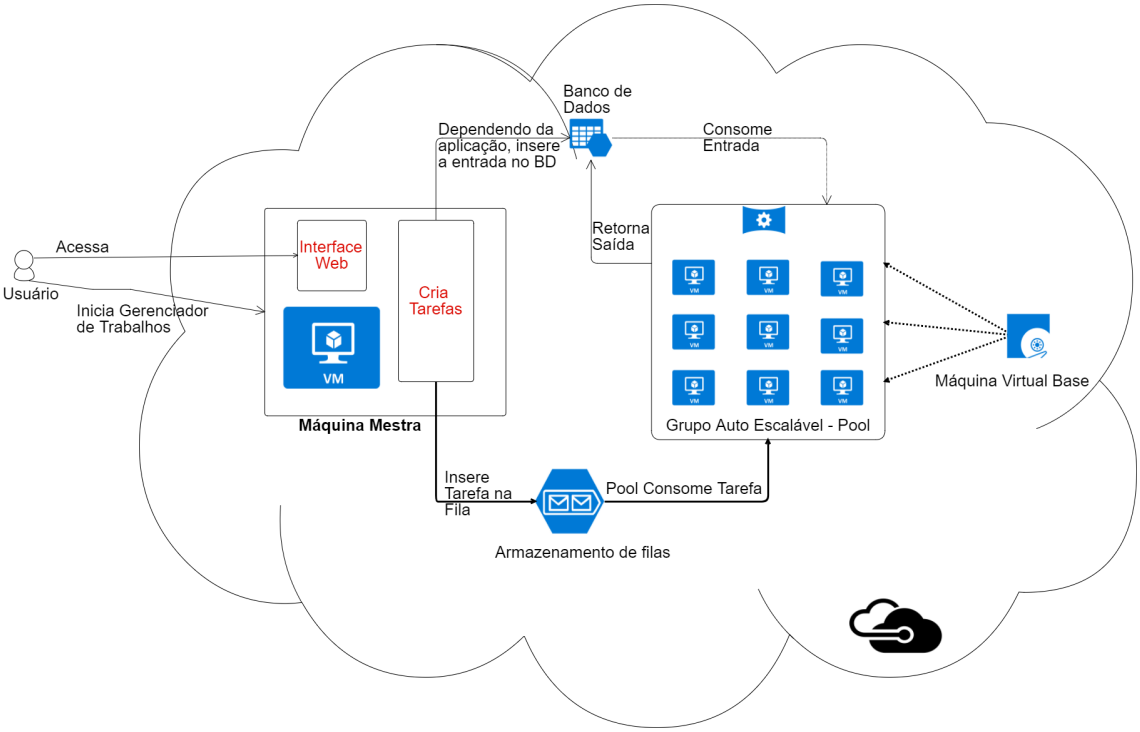

Figura 4.1: Arquitetura geral do CloudBoss.

- container do banco de dados onde será armazenado o arquivo de entrada;
- caminho do banco de dados de saída;
- nome que o Blob (arquivo) de saída terá;
- caminho para o arquivo de saída que o programa do usuário espera na máquina em que a tarefa será executada;
- caminho para o arquivo de entrada, quando necessário.

O terceiro sub serviço consiste no armazenamento de imagens de máquinas virtuais. Este é representado pelo componente "Máquina Virtual Base", situado no canto direito da Figura 4.1. Sua principal função é servir de imagem base para as máquinas virtuais do Grupo Auto Escalável. Assim sendo o usuário pode criar imagens de máquinas virtuais a partir de outras. Uma imagem deve possuir todos os programas e configurações necessárias para executar a aplicação.

Por fim temos o Grupo Auto Escalável ou Pool. No Grupo Auto Escalável teremos máquinas trabalhadoras, que consumirão tarefas da fila. Este Pool aumenta ou diminui dependendo do número de mensagens da fila. Assim, a única função de uma máquina virtual, dentro do Grupo Auto Escalável, é executar uma tarefa. Executar uma tarefa consiste nas seguintes etapas:

1. consumir uma tarefa (mensagem) da fila de tarefas;
2. realizar o *Download* do arquivo de entrada, caso necessário;

3. executar a tarefa;
4. verificar o retorno da execução;
   4.1. em caso de sucesso:
      i. enviar o arquivo de saída para o Banco de Dados;
      ii. enviar o tempo de execução para uma fila de informações;
   4.2. em caso de falha:
      i. enviar a saída do console como mensagem para uma fila de erros;
5. verificar se chegou algum sinal para a máquina ser desligada;
   5.1. caso seja necessário ser desligada:
      i. para todas as operações e avisa a máquina mestra que ela, máquina trabalhadora, já pode ser desligada;
   5.2. caso não seja necessário ser desligada;
      i. volta para a etapa 1.

4.3 Algoritmos e estratégias

Esta seção apresenta algoritmos e estratégias que foram desenvolvidas como parte do sistema proposto. Os objetivos destes algoritmos e estratégias são reduzir custos, lidar com falhas e erros de execução.

4.3.1 Estratégia de auto escalonamento

O algoritmo de auto escalonamento tem o papel de utilizar os recursos - máquinas virtuais - de maneira a evitar desperdício. Definimos desperdício como uma máquina virtual ligada e em estado ocioso. Portanto, o objetivo é manter sempre as máquinas com tarefas disponíveis para executar. Esse cenário é viabilizado pelo ajuste do número de máquinas virtuais ligadas de acordo com a taxa de chegada de novas tarefas na fila. Logo, o número de tarefas na fila precisa ser maior que o número de máquinas no *pool*. As Equação 4.1 e 4.2 são utilizadas neste trabalho para realizar tal ajuste [34]:

$$n = \min\left(\left(\lceil\frac{numberOfMessages}{tasksMachinesHandle}\rceil - staticNumberOfMachines\right), maxVM\right) \quad (4.1)$$

$$result = \max(n, 0) \quad (4.2)$$

Em que:

- **numberOfMessages** representa a quantidade de tarefas na fila;

- **tasksMachinesHandle** indica quantas tarefas simultâneas uma única máquina pode executar, para casos de máquinas com mais de um núcleo, em que cada núcleo do processador executa uma tarefa diferente;
- **staticNumberOfMachines** é o número mínimo de máquinas que devem ficar ligadas, independentemente do número de tarefas na fila e do escalonador;
- **maxVM** representa a quantidade máxima de máquinas que podem ficar ligadas.

Usando a equação 4.1, verifica-se quantas tarefas existem na fila e compara-se com a quantidade de máquinas virtuais ligadas, levando em consideração a quantidade de tarefas que cada máquina consegue executar simultaneamente. Portanto, o número de máquinas ligadas será o menor valor entre o número de máquinas necessárias para eliminar todas as tarefas e o número máximo de máquinas permitidas. A equação 4.2 garante que não teremos números negativos de máquinas No caso do sistema proposto, esse ajuste é executado a cada cinco minutos.

4.3.2 Algoritmo para otimizar o custo de processamento

A estratégia para definir o custo de uma máquina é importante para o sistema. É com ela que o sistema identifica e classifica quais máquinas possuem a melhor relação entre custo e benefício para executar a carga de trabalho do usuário.

O algoritmo desenvolvido neste trabalho, chamado de *Warmup*, executa uma pequena parte da carga de trabalho em diferentes máquinas, que chamamos de semente. O resultado final é o tempo de execução multiplicado pelo preço, por hora, de cada máquina virtual. Este resultado final é usado como indicativo da máquina virtual com menor custo para a carga de trabalho em questão. O *Warmup* é executado apenas uma vez e é uma etapa totalmente opcional.

A primeira abordagem para descobrir a melhor semente envolvia executar uma pequena carga de trabalho em todas as máquinas virtuais disponíveis. Porém isto não se mostrou eficiente, visto que o provedor de nuvem possui diversas configurações de máquinas com preços variados. A diferença de preço entre duas máquinas pode ser de dois centavos de dólares a hora até 8,69 dólares a hora. Além da diferença de valores, o alto número de configurações disponíveis aumenta o tempo e custo para esta etapa. Aproximadamente 45 configurações de máquinas virtuais na AWS e 34 no Azure - na data de elaboração deste trabalho. Com isso separou-se a etapa de testes em duas fases.

Na primeira fase as máquinas são agrupadas por semelhança. Uma máquina de cada grupo é usada para executar uma pequena parte da carga de trabalho. O resultado desta fase indica qual o melhor grupo de máquinas

Após decidir qual o melhor grupo, a mesma parcela da carga de trabalho é executada por todas as máquinas pertencentes a este grupo. Seguindo o mesmo processo, agora temos também as informações de tempo e preço das máquinas virtuais deste grupo. Logo, temos um indicativo da melhor máquina para executar nossa carga de trabalho.

Vamos utilizar a Figura 4.2 para exemplificar. Temos dois grupos de máquina. O primeiro é representado pelo grupo de máquinas que possui seu desempenho focado em

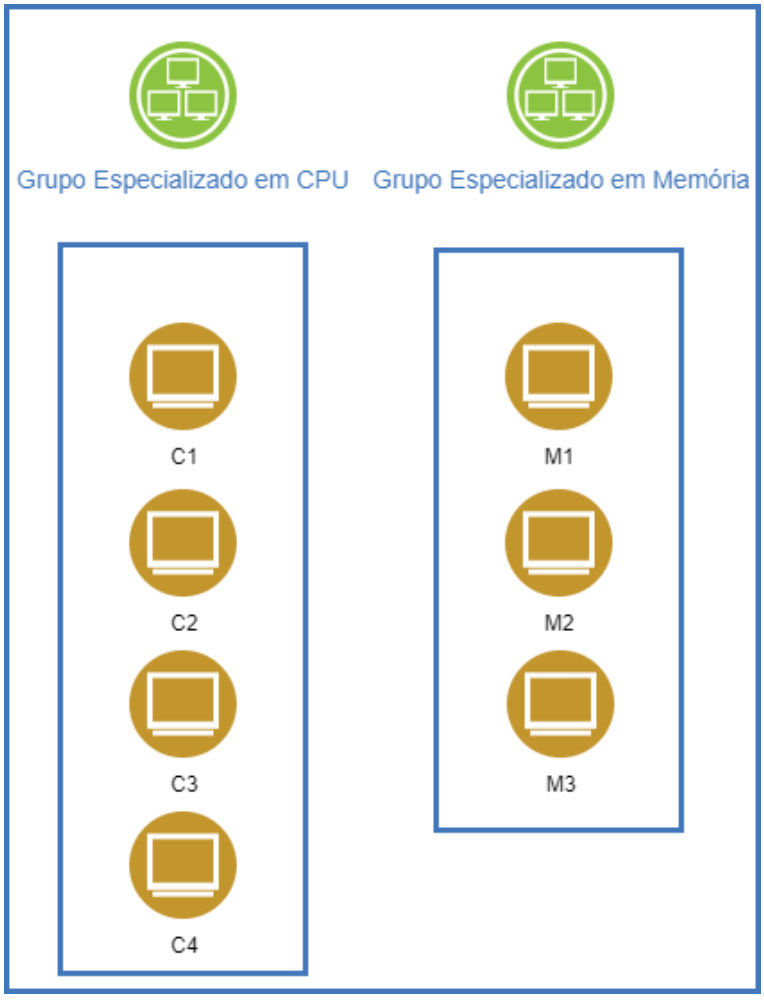

Figura 4.2: Exemplo de grupo de máquinas.

CPU. Em seu grupo estão as máquinas C1, C2, C3 e C4. O outro grupo, possui seu desempenho focado em memória e é representado pelas máquinas M1, M2 e M3. Na primeira etapa do *Warmup*, o sistema irá executar uma pequena parte da carga de trabalho na máquina C1 e na máquina M1. O resultado desta etapa indicará se a carga de trabalho tem mais afinidade com máquinas voltadas para desempenho de processamento ou de memória. Para fins de exemplo, vamos supor que o resultado foi que a máquina C1 se saiu melhor. Logo iremos executar a mesma carga de trabalho para máquinas C2, C3 e C4. Com os dados gerados podemos indicar qual a melhor máquina entre as 4 (C1, C2, C3 e C4).

4.3.3 Estratégia para lidar com falhas

Há muitos sistemas que precisam lidar com cargas de trabalho grandes, que levam dias para serem executadas. Dependendo da falha ocorrida, esta não pode atrapalhar o andamento geral da execução. Apesar do foco do trabalho não ser lidar com estas falhas, um nível mínimo de tolerância a falhas é algo desejado. Portanto, para lidar com possíveis problemas, o sistema utiliza uma biblioteca chamada Forever-service [35]. Esta biblioteca transforma uma aplicação Node.js em um serviço para o sistema operacional. Desse modo, caso ocorra uma falha, a aplicação é automaticamente reiniciada.

Aliado ao serviço, o sistema mantém um pequeno sistema de banco de dados que é usado internamente e tem seus dados descartados ao término do trabalho. Dessa forma, operações críticas tem suas informações armazenadas, tais como: número de máquinas ligadas, tempos de execução entre outras. Quando o sistema é reiniciado devido a uma falha, os dados críticos podem ser recuperados.

4.3.4 Estratégia para lidar com erros de execução

Além dos erros comuns do sistema operacional e de *Hardware*, podem aparecer erros de execução durante a execução da carga de trabalho. Erros de execução podem ocorrer por diversos motivos, são exemplos: falta de memória RAM e atraso em alguma comunicação. Para tratar essas falhas, o sistema executa um fluxo dividido em duas etapas.

A primeira etapa simplesmente executa toda a carga de trabalho em máquinas virtuais de mesma configuração. Essa configuração de máquina pode ter sido escolhida automaticamente pelo sistema, usando o *Warmup* da seção 4.3.2, ou escolhida manualmente pelo usuário. Caso seja detectado algum erro durante a execução, o sistema envia aquela tarefa para uma fila de erros. O erro é detectado quando a aplicação que está sendo executada possui um código de erro diferente de 0 (zero).

Ao término da execução da primeira etapa, o sistema verifica se há tarefas na fila de erro. Caso não haja, o sistema finaliza as operações. Caso contrário, o sistema elenca uma nova configuração de máquina para executar as tarefas pendentes, normalmente uma mais potente. Na sequência, começa a transferir as tarefas da fila de erros para a fila principal. Assim como na primeira etapa, caso seja detectado algum erro, o sistema envia essa tarefa novamente para a fila de erros. No entanto, ao término da execução o sistema apenas informa o usuário o fim da execução da carga de trabalho. Caso ainda possua tarefas na

fila de erros, estas são apenas reportadas ao usuário.

4.4 Considerações

Toda a infra estrutura desenvolvida neste trabalho é distribuída. Logo, abre a possibilidade para máquinas fora do Azure servirem como máquinas trabalhadoras. Assim, caso o usuário possua máquinas disponíveis em servidores locais, ou em outros provedores de nuvem, estas poderiam ser usadas para consumir tarefas. Neste caso, haveria duas possibilidades de integração.

Na primeira possibilidade, a máquina mestra apenas teria a informação de que há mais máquinas ligadas, mas não possuiria o controle sobre elas. Para fazer isso, o sistema precisaria de uma adaptação na etapa em que recebe os resultados de uma máquina. Outra adaptação seria necessária no *software* que executa nas máquinas trabalhadoras, pois não poderiam consumir tarefas de filas que orquestram quais máquinas devem estar ligadas e desligadas.

Um segundo caso seria uma modificação mais profunda e complexa do sistema. Nela o sistema poderia gerenciar também máquinas de outros provedores e de servidores privados do usuário. O sistema precisaria abstrair diversas operações para poder funcionar da maneira correta. Nenhuma destas modificações foram implementadas por não ser o foco deste trabalho, mas são sugeridas como trabalhos futuros.

Capítulo 5

Casos de uso e resultados

Para a validação da infraestrutura, foi realizado um conjunto de testes. Neste capítulo serão apresentados e discutidos os mais importantes, suas metodologias, objetivos e resultados.

5.1 Ligamento e desligamento de máquinas virtuais

Uma das etapas importantes do Cloudboss é a auto escalabilidade. Por meio da escalabilidade mantemos somente o número de máquinas necessárias para lidar com a carga de trabalho do usuário. Para manter sempre o número ideal, o sistema precisa periodicamente ligar e desligar máquinas.

Comumente, a cobrança de uma máquina virtual na nuvem pública é baseada em seu tempo de uso. Neste caso, o tempo gasto para a máquina ser criada, ligada, desligada e destruída também é contabilizado. Logo, um sistema de auto escalabilidade precisa estar ciente dos impactos que a manipulação do estado destas máquinas causa. O experimento apresentado nesta seção tem como finalidade entender qual é este impacto.

5.1.1 Metodologia

Para realizar o experimento foi definida uma estratégia para criar, desligar e destruir máquinas. Estes comandos foram disparados por uma máquina virtual com sistema operacional Linux Ubuntu [36], versão 14.04.02. Para tanto, foi criado um *script* que, por meio de chamadas REST, criava e apagava máquinas virtuais. Foram escolhidas quatro configurações diferentes para as máquinas virtuais, sendo elas:

- A1 - Máquina de propósito geral com um núcleo, 1,75GB de memória RAM e 70 GB de espaço em disco rígido;
- A3 - Máquina de propósito geral com quatro núcleos, 7GB de memória RAM e 285 GB de espaço em disco rígido;
- A10 - Máquina especializada em CPU com oito núcleos, 56GB de memória RAM e 382 GB de espaço em disco rígido;

- D3 - Máquina de propósito geral com quatro núcleos, 14GB de memória RAM e 200GB de espaço em disco de estado sólido.

Para compor o sistema operacional das máquinas virtuais foram escolhidos três sistemas - Ubuntu, OpenSuse e Windows Server. O objetivo de realizar o teste com três sistemas operacionais é de verificar se o sistema operacional pode influenciar no processo de criar e apagar máquinas.

Além do fator do sistema operacional, foi verificado se o provedor de nuvem poderia fazer algum *cache* da máquina, fazendo com que criações consecutivas de máquinas com mesma configuração fossem mais rápidas. Em resumo, foram testados três sistemas operacionais em quatro configurações de máquinas diferentes; primeiro criando e desligando as máquinas de uma mesma configuração de forma sequencial e, depois, intercalando entre diferentes configurações de máquina virtual.

5.1.2 Resultado e conclusão

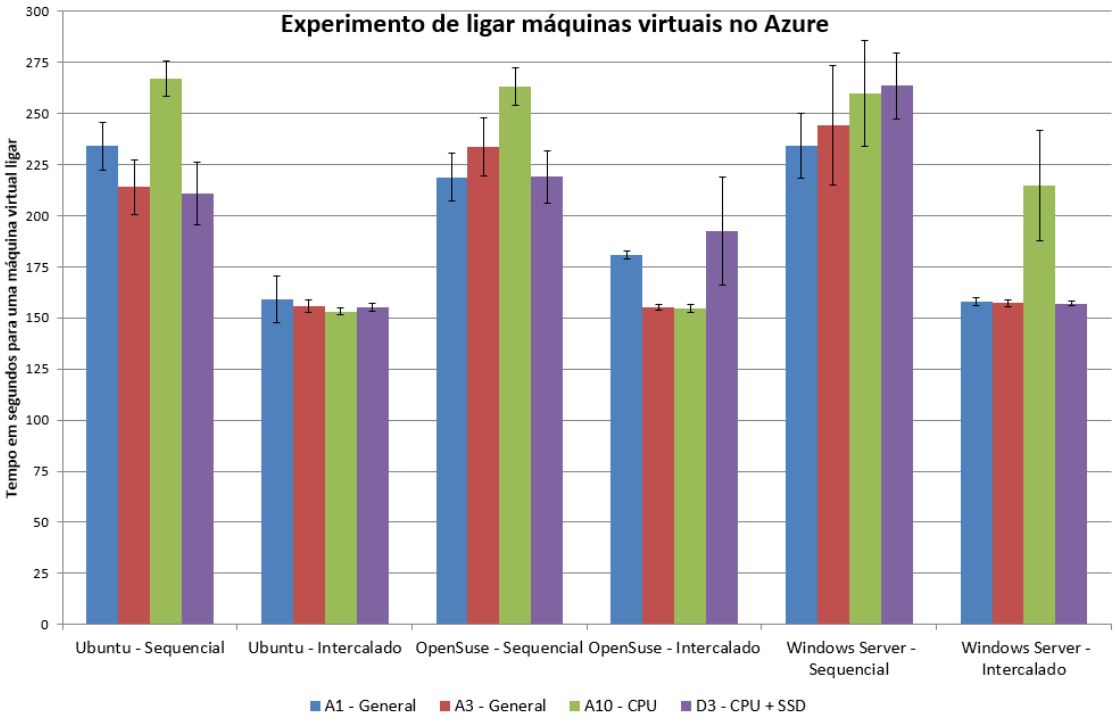

Figura 5.1: Resultado do experimento de ligar máquina virtuais.

A Figura 5.1 sinteriza os resultados alcançados com o experimento. Podemos observar que o sistema operacional não influencia significativamente na diferença de tempo entre criar a máquina e a máquina estar disponível para uso. Pode-se notar também que a diferença entre máquinas virtuais de diferentes configurações não afetou o tempo para criação das máquinas, sendo essa diferença de no máximo 57 segundos no caso do Windows Server - Intercalado, valor baixo se comparado a cargas de trabalho que duram dias. A hipótese é de que, apesar do processo de inicialização do sistema operacional ser mais rápido nas máquinas virtuais mais potentes utilizadas neste teste, a alocação física destas máquinas

seria mais demorada. Para validar esta hipótese seria necessário ter acesso a dados internos do funcionamento do provedor da nuvem, como o virtualizador e escalonador que utilizam e a política interna de alocação física.

É interessante notar também que em todos os casos a criação intercalada foi mais rápida do que a sequencial. Não foi possível concluir o motivo deste comportamento, pois também depende de dados internos do provedor de nuvem. As medidas coletadas de tempo para desligar e desalocar as máquinas foram iguais, independente da máquina e abordagem escolhida.

Com isso, concluímos que a configuração da máquina virtual não irá impactar significativamente no tempo e custo total no momento de criação das máquinas. Podemos concluir também que o tempo médio para ligar uma máquina virtual e ter ela disponível varia entre dois e quatro minutos. Portanto, a criação do auto-escalonador foi feita sem atribuir qualquer peso de decisão sobre a configuração da máquina em questão.

5.2 Execução do Parsec

Como já mencionado, os provedores de nuvem possuem diferentes configurações de máquinas virtuais com diversos preços. Portanto, é de se esperar que uma aplicação tenha tempo de execução diferente em máquinas com configurações distintas. Esta diferença de tempo pode impactar diretamente no preço final de uma carga de trabalho, principalmente, se esta for muito grande.

Exemplificando, podemos ter uma aplicação que faz muitos acessos a disco demandando 1 hora de execução, a 50 centavos a hora, em uma máquina de propósito geral e 30 minutos de execução, a 75 centavos a hora, em uma máquina com SSD. Neste caso, a máquina de maior custo por hora resultou em um custo total mais baixo para o usuário[1].

Outro caso também possível é ter uma máquina de menor custo que leva bem mais tempo para executar a carga de trabalho, porém que no final ainda custe menos do que uma máquina mais cara e mais rápida. O cenário que representa esse caso é o de uma aplicação *CPU-bound*. Executando-a em uma máquina com muitos núcleos a aplicação pode demorar uma hora para terminar e custar 1 real por hora. Esta mesma aplicação, em uma máquina mais simples, pode levar quatro horas, porém a 20 centavos a hora. Aqui, apesar do tempo elevado, a máquina mais simples foi mais barata.

Escalando o último cenário, podemos ter várias máquinas tratando várias instâncias do problema. Portanto, o tempo de execução é reduzido e o custo benefício se mantem. Desse modo, o principal objetivo deste experimento é identificar se aplicações com necessidades semelhantes irão ter a mesma preferência de máquinas, levando como critério a relação entre custo e benefício. Vale ressaltar que quanto melhor a utilização dos recursos na nuvem, menor será o gasto operacional.

[1] O preço em hora no Microsoft Azure é para referência, pois nesse caso o provedor realiza sua cobrança por minutos.

Tabela 5.1: Subconjunto SPLASH-2 da suíte do PARSEC [3].

Programas	**Domínio da Aplicação**	**Estrada utilizada**
barnes	Computação de Alto Desempenho	65.536 partículas
cholesky	Computação de Alto Desempenho	tk29.O
fft	Processamento de Sinais	4.194.304 pontos
fmm	Computação de Alto Desempenho	65.536 partículas
lu	Computação de Alto Desempenho	1024x1024 matriz, 64x64 blocos
ocean	Computação de Alto Desempenho	514x514 grade
radiosity	Gráficos	Quarto grande
radix	Geral	8.388.608 Inteiro
raytrace	Gráficos	Carro
volrend	Gráficos	cabeça
water	Computação de Alto Desempenho	4.096 moléculas

5.2.1 PARSEC

Princeton Application Repository for Shared-Memory (PARSEC - do inglês - Repositório de Aplicações de Princeton para Computadores com Memória Compartilhada) é uma suíte de *benchmarks*, voltada para programas *multithreaded*. Como visto na Tabela 5.1, a suíte é extensa e cobre diversos problemas [3].

A criação do PARSEC visa atender cinco características, são elas:

- ***Multithreaded*:** visa atender as necessidades de computadores modernos que possuem mais de um núcleo de processamento. Assim o programa em execução consegue aproveitar melhor os recursos disponíveis na máquina;
- **Ferramentas Emergentes:** apresentar ferramentas que, embora ainda desconhecidas, acredita-se que irão se popularizar em breve;
- **Diversidade**: PARSEC possui uma vasta coleção de aplicações de várias áreas e necessidades;
- **Não é focada somente em Computação de Alto Desempenho:** apesar de computação de alto desempenho possuir altos níveis de consumo de processamento, estas são apenas uma pequena parcela do conjunto de aplicações que demandam muita computação. É cada vez mais comum aplicações utilizando muita computação com vários núcleos trabalhando em paralelo;
- **Pesquisa:** o PARSEC é focado em auxiliar pesquisas.

O PARSEC foi utilizado neste trabalho de mestrado por já ter muitas pesquisas explorando seus diversos aspectos, bem como a uma grande variedade de *benchmarks* disponibilizados.

5.2.2 Metodologia

O primeiro passo para o experimento foi descobrir qual o tempo médio de execução de cada programa da suíte do PARSEC em cada configuração de máquina disponível no

Azure. Cada aplicação foi executada 30 vezes em cada uma das máquinas disponíveis. Após a conclusão das execuções, foi multiplicado o tempo médio pelo valor da hora da máquina.

Neste experimento foi coletado o tempo de execução, descartando-se qualquer outro tempo, como por exemplo o tempo de ligar e desligar a máquina. Com o tempo de execução, foi gerado o custo para executar as aplicações.

5.2.3 Resultado e conclusão

Para cada um dos *benchmarks* foi traçada a curva de Pareto que indica o conjunto de máquinas que fornece o melhor compromisso entre custo e tempo de execução. As Figuras 5.2 e 5.3 apresentam os resultados obtidos para os *benchmarks* radix e lu, respectivamente [2]. Os gráficos omitem a identificação de algumas das máquinas. Com base nos resultados é possível verificar que as máquinas mais baratas levam mais tempo para executar a carga de trabalho e máquinas que executam a aplicação mais rapidamente, custam mais. Em alguns cenários, o tempo de execução pode ser reduzido se utilizarmos mais de uma máquina; assim, o custo é mantido abaixo do que uma outra máquina com tempo de execução similar. Porém, os provedores de nuvem limitam a quantidade de máquinas simultâneas em uma mesma região. Logo, essa limitação pode prejudicar a velocidade com que a carga de trabalho será concluída. Desse modo, é importante levar em consideração a relação custo e tempo de execução das máquinas no caso do tempo de execução ser uma restrição.

Os resultados coletados ainda permitiram observar dois padrões entre as aplicações testadas. No primeiro padrão (Categoria A), as máquinas da família DV2, que são máquinas com SSD e configurações balanceadas, apareceram no limiar da curva pareto. A presença delas é um indicativo do custo reduzido destas máquinas demonstrando um equilíbrio entre as operações realizadas pelos *benchmarks* desta categoria. O segundo padrão (Categoria B) é marcado pela presença de máquinas A10 e A11. Essas máquinas possuem maior poder computacional, logo mesmo estas máquinas sendo mais caras que as máquinas de propósito geral, elas apresentaram uma boa relação de tempo e custo de execução. Mais curvas paretos podem ser visualizadas no Apêndice A.

A categorização geral das máquinas está sumarizada na Tabela 5.2. Nela pode-se observar que somente cargas de trabalho de computação de alto desempenho e uso intensivo estão na categoria B. Já a categoria A possui diversos tipos de aplicações. Nota-se também que em algumas cargas de trabalho ocorreram erros durante a execução. Para o caso do *cholesky* e *volrend* ocorreram erros durante a instalação e configuração das aplicações. O *fft* executou em poucas máquinas, pois necessita de muita memória para poder ser executado, logo foi ignorado.

Adicionalmente, os resultados obtidos serviram de guia para categorizar e agrupar as máquinas utilizadas na etapa de *Warmup*, discutido na subseção 4.3.2. Características específicas de configuração, como máquinas especializadas em redes, também foram levadas em consideração na categorização. A Figura 5.4 mostra como ficou a divisão em grupos. As máquinas marcadas em azul indicam aquelas que serão utilizadas na primeira etapa do *Warmup*. Note que para dois dos grupos duas máquinas foram escolhidas para participar

[2] As curvas de Pareto para o restante dos *benchmarks* estão disponíveis no Apêndice A.

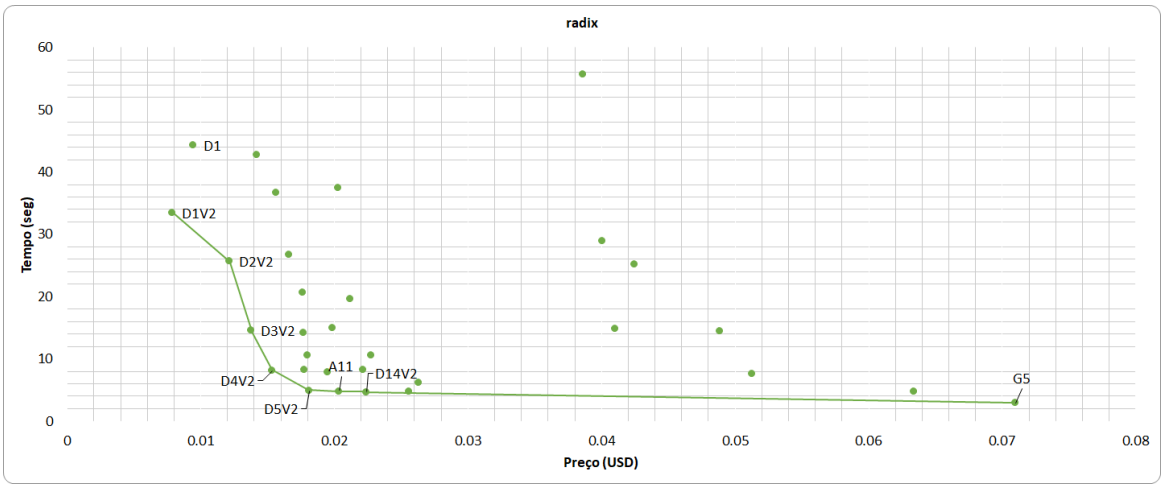

Figura 5.2: Curva pareto para *benchmark* radix. Categoria A.

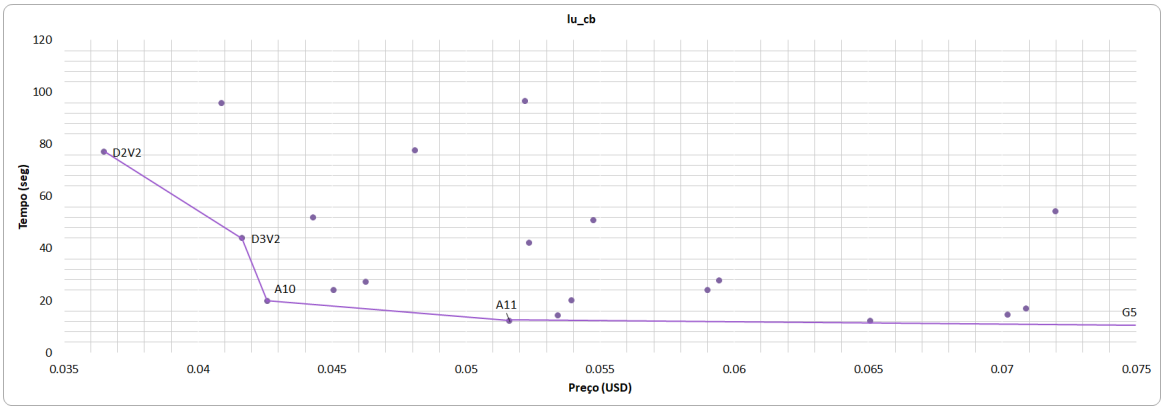

Figura 5.3: Curva pareto para *benchmark* lu_cb. Categoria B.

Tabela 5.2: Categorização dos *benchmarcks*.

Programas	**Domínio da Aplicação**	**Categoria**
barnes	Computação de Alto Desempenho	A
cholesky	Computação de Alto Desempenho	Erros durante execução
fft	Processamento de Sinais	Erros em algumas máquinas
fmm	Computação de Alto Desempenho	A
lu_cb	Computação de Alto Desempenho	B
lu_ncb	Computação de Alto Desempenho	A
ocean_cp	Computação de Alto Desempenho	B
ocean_ncp	Computação de Alto Desempenho	B
radiosity	Gráficos	A
radix	Geral	A
raytrace	Gráficos	A
volrend	Gráficos	Erros durante execução
water	Computação de Alto Desempenho	A

da primeira etapa. Esta escolha se deu pelo fato destas quatro máquinas retornarem bons resultados para a maior parte dos *benchmarks* testados.

5.3 Execução de carga de trabalho no Cloudboss

Este experimento tem o intuito de validar a metodologia proposta no Cloudboss para escolha da melhor máquina para executar uma determinada carga de trabalho. Essa funcionalidade está escrita na subseção 4.3.2.

5.3.1 Metodologia

Para atingir os objetivos deste experimento, foi executada uma carga de trabalho com 1000 simulações para o *benchmark* radix. Esta execução ocorreu em todas as máquinas disponíveis na plataforma Azure, na data de execução do experimento. Assim, foi possível descobrir em quais máquinas a execução de toda a carga de trabalho foi a mais barata.

Além da execução da carga de trabalho, foi executado o algoritmo que define a melhor máquina para executar a carga de trabalho - *Warmup*. Para tal, o *Warmup* foi executado levando em consideração 10, 50 e 100 entradas para as simulações para comparar se a precisão do algoritmo se altera. Com os dados retornados pelo *Warmup* e os dados coletados na execução da carga de trabalho, foi gerado o custo para executar uma instância da carga de trabalho em uma máquina específica. É importante notar que, o *Warmup* descrito não é executado em todas as máquinas de uma só vez, como já descrito na subseção 4.3.2.

5.3.2 Resultados e conclusões

Os dados coletados nos experimentos foram normalizados para simplificar a visualização dos resultados. Assim, o menor valor em dólares para a execução recebeu o valor 1

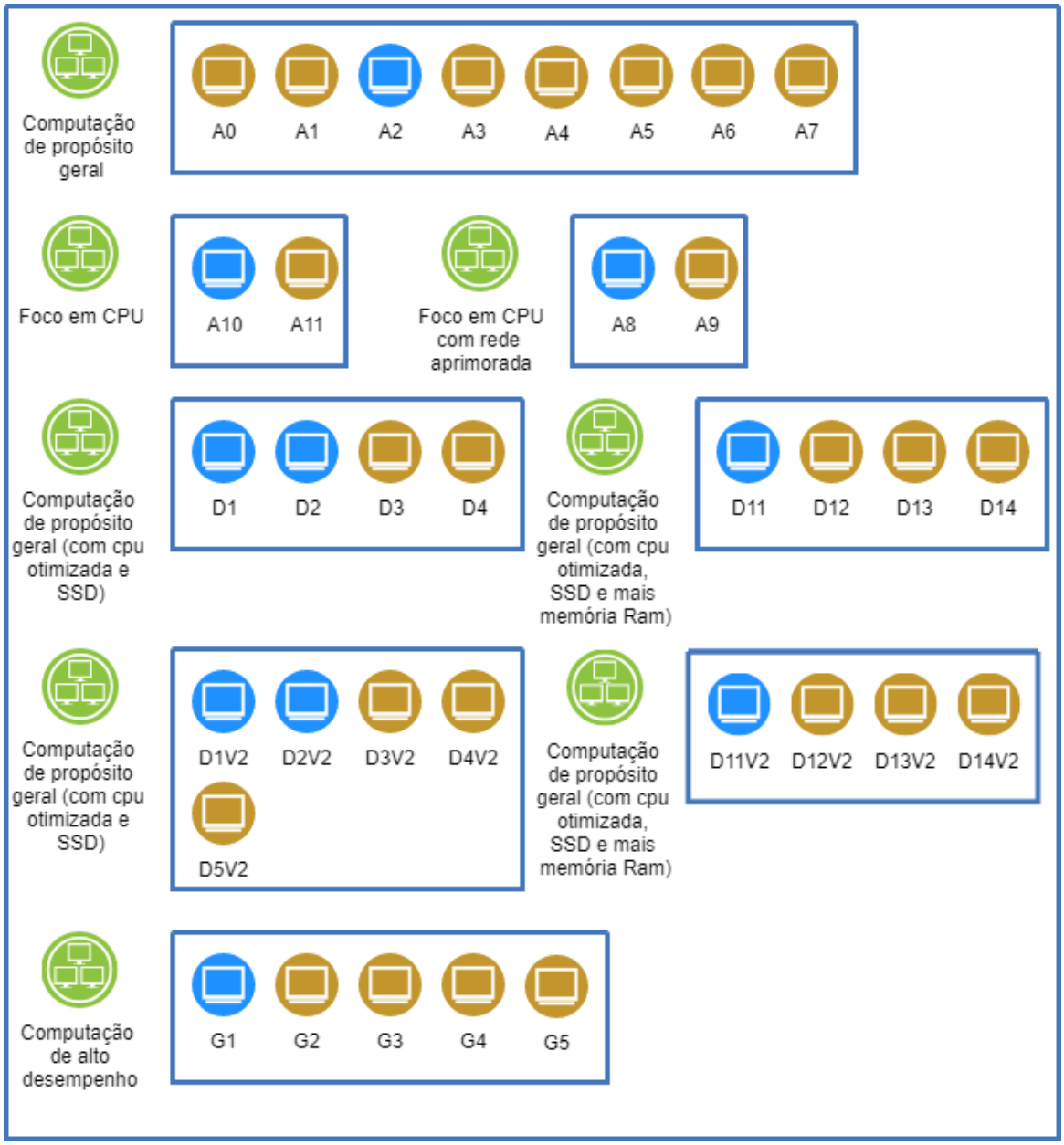

Figura 5.4: Máquinas virtuais divididas em grupos para o *Warmup*.

(um) e os demais foram representados com o seu valor de custo dividido pelo menor valor. Os valores de custo obtidos a partir das 1000 simulações foram chamados de real. Os demais valores do *Warmup* foram chamado de *Warmup* e um sufixo numérico que representa a quantidade de entradas utilizadas no processo para descobrir a máquina ideal. É importante notar que o sistema de *Warmup* é executado em duas etapas, como já descrito na subseção 4.3.2. Com os dados normalizados, foi possível desenhar o gráfico que representa a primeira etapa do *Warmup* (Figura 5.5) e o gráfico que representa a segunda etapa, Figura 5.6. Nessas Figuras o eixo horizontal representa o valor normalizado e o eixo vertical representa o modelo de máquina que fez a execução.

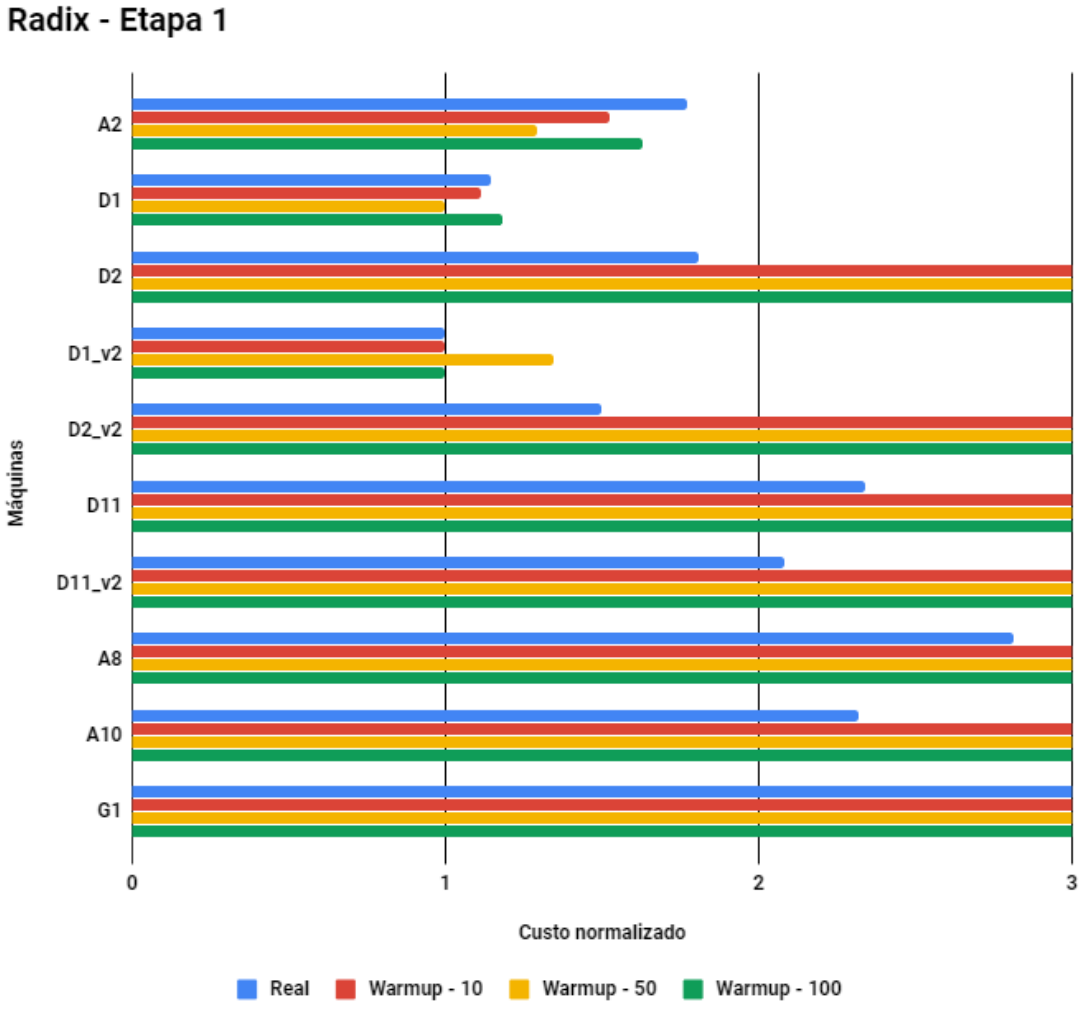

Figura 5.5: Resultado da primeira etapa do *Warmup* para o radix.

Para o caso do Radix, Figuras 5.5 e 5.6, pode-se observar que o *Warmup* acertou duas vezes qual a máquina com menor custo (Máquina D1_v2, *Warmup*-10 e *Warmup*-100), e no caso em que errou (Máquina D1, *Warmup*-50), esta mesma é uma máquina apenas 15% mais cara que a ideal[3].

[3]No caso de escolher a máquina D1 na primeira etapa, as máquinas D3 e D4 seriam testadas na segunda etapa, mas seriam descartadas por resultarem em custo maior. O mesmo aconteceria para as máquinas que estão no grupo da máquina D1_V2

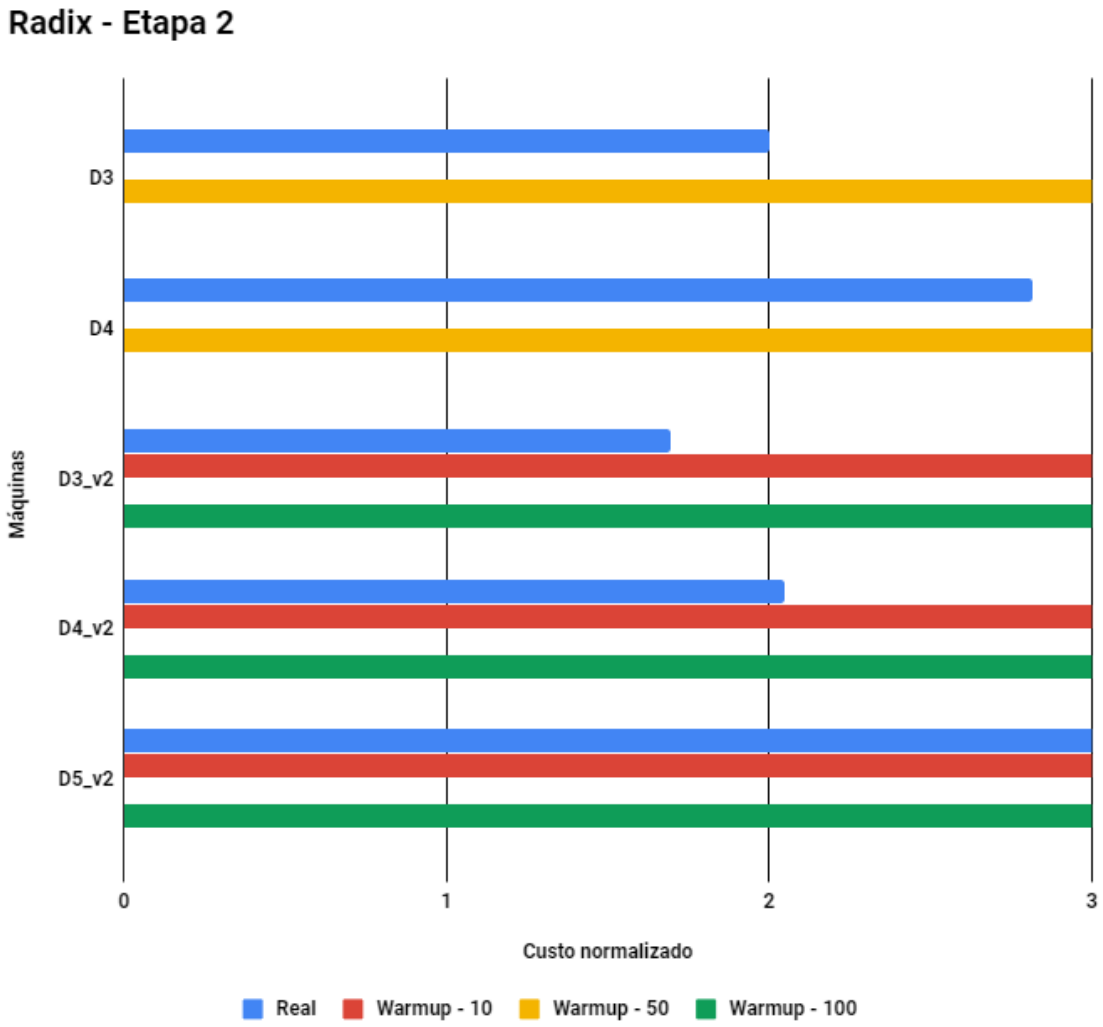

Figura 5.6: Resultado da segunda etapa do *Warmup* para o radix.

Portanto, pode-se dizer que para este caso o sistema faria uma boa escolha ao auxiliar o usuário na escolha das máquinas, independente do número de entradas utilizadas para *Warmup*. É importante ressaltar também a coerência dos dados coletados neste experimento com a curva pareto desenhada para o radix na Figura 5.2

5.4 Execução do axisimétrico

O axisimétrico é um problema da engenharia de petróleo, cujo objetivo é calcular a vazão e pressão do poço de petróleo para dadas condições do reservatório e poço. Geralmente, é necessário calcular diversas vezes o problema, principalmente para análise de incertezas, ou mesmo em fase de projeto, em que se busca um projeto ótimo do poço. Deixar as equações escritas de forma axissimétrica (e por consequência 2D) traz ganho computacional se comparado à formulação 3D. O código axisimético executado neste experimento foi o disponibilizado pela biblioteca NeoPz [37].

5.4.1 Metodologia

A metodologia utilizada neste experimento foi muito similar à utilizada na seção 5.3. A maior diferença entre os experimentos foi que, neste caso, a carga de trabalho foi estipulada em 100.000 tarefas. Há também a diferença entre as tarefas em si, pois por se tratar de um problema real de simulação, foi possível gerar 100.000 entradas diferentes e aleatórias para compor a carga de trabalho.

Com isso a carga de trabalho foi executada em todas as máquinas disponíveis e o custo em cada máquina foi coletado.

Após a execução da carga de trabalho, foi executado o algoritmo de *Warmup* com 10, 50 e 100 entradas e os dados de custo foram coletados. Os dados foram normalizados seguindo o padrão da seção 5.3.

5.4.2 Resultado e conclusão

Os dados do experimento estão sumarizados nas Figuras 5.7 e 5.8. Nessas Figuras podemos observar um resultado semelhante ao obtido com o *benchmark* Radix. O resultado das etapas de *Warmup* estão bem próximos do resultado final, com diferença média do resultado real de aproximadamente 15%. Consideramos este um resultado positivo, visto que mesmo com essa diferença, o sistema conseguiu indicar a melhor máquina para executar a carga de trabalho - Máquina D1_v2.

Outro resultado importante é a economia de se realizar o *Warmup* em duas etapas. Para o caso deste experimento, caso realizássemos o teste em todas as máquinas, teríamos um custo para o *Warmup* de 100 entradas de aproximadamente 138 dólares. Para o caso em que dividimos o teste em duas etapas, utilizando as máquinas das Figuras 5.7 e 5.8, temos o custo aproximado de 27 dólares. Isso representa uma economia de aproximados 111 dólares ou uma razão de 5,1.

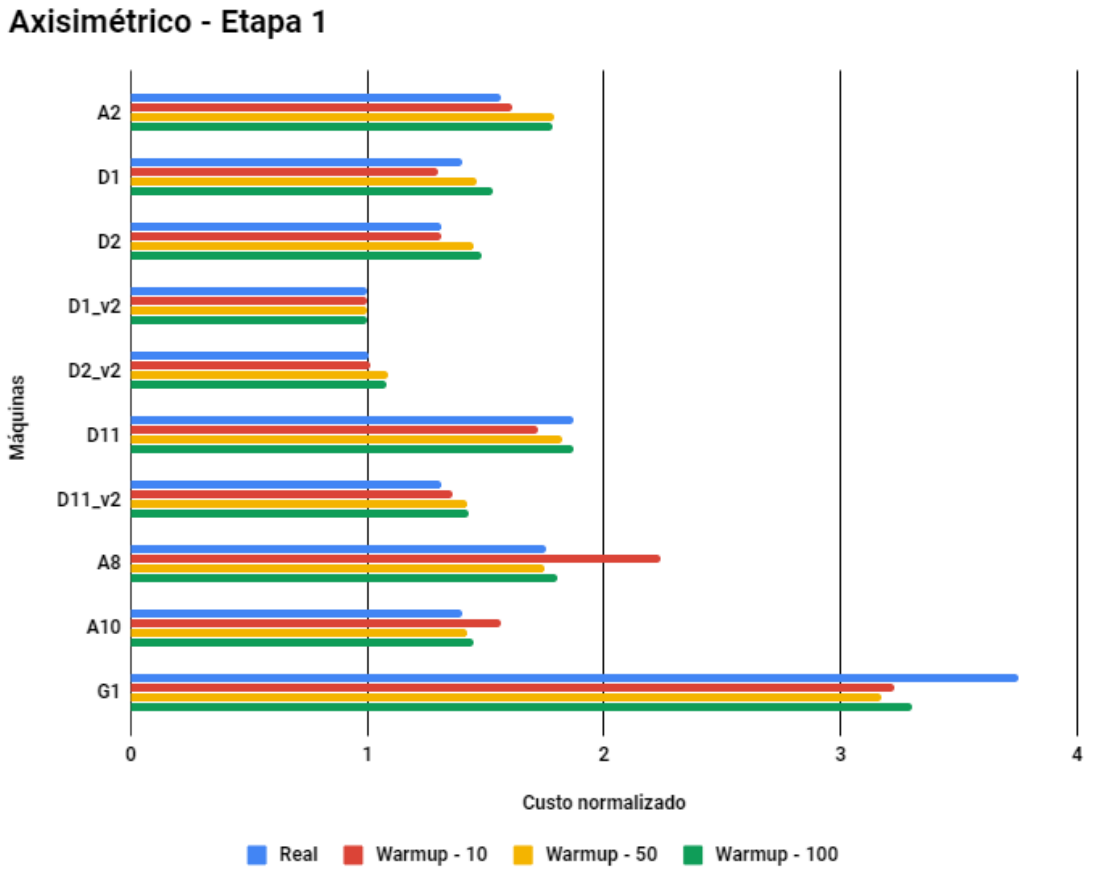

Figura 5.7: Resultado da primeira etapa do *Warmup* para o axisimétrico.

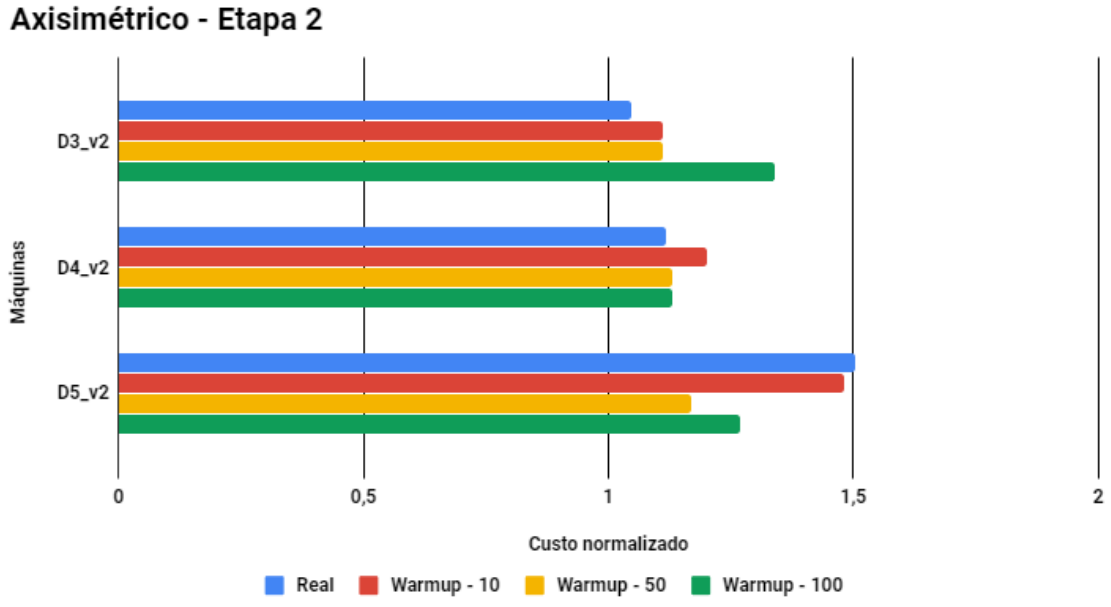

Figura 5.8: Resultado da segunda etapa do *Warmup* para o axisimétrico.

Capítulo 6

Conclusões

Computação em nuvem tem mudado como a indústria e a academia utilizam seus recursos quando o assunto é infraestrutura de máquinas. Aplicações de alto desempenho, como simulações, demandam alto poder computacional para que seja possível obter seus resultados em um tempo aceitável. Para criar uma estrutura local para atender estes requisitos, é necessário conhecimento em montagem de *clusters*, configurações de plataformas de execução distribuída e altos investimentos iniciais em equipamentos, que nem sempre serão utilizados.

Com o trabalho desenvolvido nesta dissertação, mostramos que é possível otimizar o custo de execução de cargas de trabalho idempotentes e independentes no provedor de nuvem da Microsoft, o Azure. Sendo as principais contribuições deste trabalho:

- levantamento do estado da arte em ferramentas para execução de simulações computacionais na nuvem;
- criação de um sistema auto-escalável para execução automática de simulações computacionais idempotentes na nuvem;
- desenvolvimento de algoritmo que indica a máquina com melhor custo benefício para a carga de trabalho do usuário - *Warmup*.

O sistema desenvolvida buscou ser simples de ser instalada e configurada. Desse modo, mesmo usuários sem muito conhecimento em infraestrutura poderão utilizá-la. Como forma de manter o controle dos custos do usuário, o sistema também se preocupa em realizar o escalonamento automático das máquinas no provedor de nuvem. Isso tudo baseado na quantidade de tarefas a serem executadas e na quantidade máxima de máquinas que o usuário está disposto a utilizar simultaneamente.

Outro aspecto importante de ressaltar é a tolerância a falhas. Por se tratar de problemas com milhares de entradas, é fundamental que o sistema se recupere após a ocorrência de algum problema. Para tal, foram utilizados recursos do provedor de serviços na nuvem, como banco de dados para armazenar o estado atual e serviços de filas de tarefas.

O *Warmup* - algoritmo de definição de melhor máquina - demonstrou boa acurácia ao indicar as melhores máquinas para executar as cargas de trabalho. A primeira ideia para o algoritmo era simplesmente testar em todas as máquinas a carga de trabalho. Porém, essa abordagem poderia ser demorada, além de custosa. Diante do problema procuramos

classificar as máquinas disponíveis em grupos para reduzir a quantidade de máquinas testadas nessa etapa. Logo, conseguimos reduzir custos na etapa de encontrar a máquina ideal e na execução geral da carga de trabalho, visto que conseguimos executar a carga de trabalho com uma máquina que melhor se adapta às necessidades da carga de trabalho.

6.1 Trabalhos futuros

O sistema foi desenvolvido utilizando recursos semelhantes em diversos provedores de nuvem. Portanto, como sugestão, recomenda-se estender o desenvolvimento do sistema para atender outros provedores de nuvem, como Amazon Web Services e Google Cloud. Assim, o usuário poderia escolher o provedor que possui maior afinidade ou contrato.

O desenvolvimento do algoritmo de *Warmup* deste trabalho utilizou como premissa o fato de que o provedor de nuvem utilizado faz a cobrança baseada em minutos, diferente de outros que cobram por hora cheia. Logo, um estudo precisa ser feito para atender esses provedores.

Outra extensão interessante para este trabalho seria dar a possibilidade do usuário definir se a escolha da melhor máquina se daria baseado em custo ou baseado em tempo de execução.

Referências Bibliográficas

[1] Buyya, R., C. S. Yeo, S. Venugopal, J. Broberg e I. Brandic: *Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility*. Future Generation Computer Systems, 25(6):599 – 616, 2009.

[2] Liu, X., Q. He, X. Qiu, B. Chen e K. Huang: *Cloud-based computer simulation: Towards planting existing simulation software into the cloud*. Simulation Modelling Practice and Theory, 26(0):135 – 150, 2012.

[3] Bienia, Christian, Sanjeev Kumar e Kai Li: *PARSEC vs. SPLASH-2: A quantitative comparison of two multithreaded benchmark suites on chip-multiprocessors*. Em *Workload Characterization, 2008. IISWC 2008. IEEE International Symposium on*, páginas 47–56. IEEE, 2008.

[4] Pan, Q., J. Pan e C. Wang: *Simulation in Cloud Computing Envrionment*. Service Sciences, International Conference on, 0:107–112, 2013.

[5] Johnson, H. E. e A. Tolk: *Evaluating the Applicability of Cloud Computing Enterprises in Support of the Next Generation of Modeling and Simulation Architectures*. Em *Proceedings of the Military Modeling & Simulation Symposium*, MMS '13, páginas 4:1–4:8, San Diego, CA, USA, 2013. Society for Computer Simulation International.

[6] Richardson, Leonard, Mike Amundsen e Sam Ruby: *RESTful Web APIs: Services for a Changing World*. "O'Reilly Media, Inc.
, 2013.

[7] Vaquero, L. M., L. Rodero-Merino, J. Caceres e M. Lindner: *A Break in the Clouds: Towards a Cloud Definition*. SIGCOMM Comput. Commun. Rev., 39(1):50–55, dezembro 2008.

[8] Juve, G., E. Deelman, K. Vahi, G. Mehta, B. Berriman, B.P. Berman e P. Maechling: *Scientific workflow applications on Amazon EC2*. Em *E-Science Workshops, 2009 5th IEEE International Conference on*, páginas 59–66, Dec 2009.

[9] Mell, Peter e Tim Grance: *The NIST definition of cloud computing*. 2011.

[10] Hwang, K. e D. Li: *Trusted Cloud Computing with Secure Resources and Data Coloring*. Internet Computing, IEEE, 14(5):14–22, Sept 2010.

[11] Jha, Shantenu, Daniel S. Katz, Andre Luckow, Andre Merzky e Katerina Stamou: *Understanding Scientific Applications for Cloud Environments*, páginas 345–371. John Wiley & Sons, Inc., 2011.

[12] Amazon: *Amazon Web Services*. http://aws.amazon.com/.

[13] Microsoft: *Microsoft Azure*. http://azure.microsoft.com/.

[14] Peng, J., X. Zhang, Z. Lei, B. Zhang, W. Zhang e Q. Li: *Comparison of Several Cloud Computing Platforms*. Em *Information Science and Engineering (ISISE), 2009 Second International Symposium on*, páginas 23–27, Dec 2009.

[15] Eucalyptus: *Eucalyptus - Open Source Private Cloud Software*. https://www.eucalyptus.com/.

[16] OpenNebula: *OpenNebula | Flexible Enterprise Cloud Made Simple*. http://opennebula.org/.

[17] OpenStack: *OpenStack - Cloud Software*. https://www.openstack.org/.

[18] Laszewski, G. von, J. Diaz, F. Wang e G. C. Fox: *Comparison of Multiple Cloud Frameworks*. Em *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, páginas 734–741, June 2012.

[19] Khan, I., H. Rehman e Z. Anwar: *Design and Deployment of a Trusted Eucalyptus Cloud*. Em *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, páginas 380–387, July 2011.

[20] Sempolinski, P. e D. Thain: *A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus*. Em *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, páginas 417–426, Nov 2010.

[21] Jorissen, K., F.D. Vila e J.J. Rehr: *A high performance scientific cloud computing environment for materials simulations*. Computer Physics Communications, 183(9):1911 – 1919, 2012.

[22] Angeli, D. e E. Masala: *A cost-effective cloud computing framework for accelerating multimedia communication simulations*. Journal of Parallel and Distributed Computing, 72(10):1373 – 1385, 2012.

[23] Jakovits, P., S. N. Srirama e I. Kromonov: *Stratus: A Distributed Computing Framework for Scientific Simulations on the Cloud*. Em *High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on*, páginas 1053–1059, June 2012.

[24] Oxford: *The Oxford BSP Toolset*. http://www.bsp-worldwide.org/implmnts/oxtool/.

[25] Sadooghi, Iman, Jesús Hernández Martin, Tonglin Li, Kevin Brandstatter, Ketan Maheshwari, Tiago Pais Pitta de Lacerda Ruivo, Gabriele Garzoglio, Steven Timm, Yong Zhao e Ioan Raicu: *Understanding the performance and potential of cloud computing for scientific applications*. IEEE Transactions on Cloud Computing, 5(2):358–371, 2017.

[26] Departamento Americano de Energia: *Fermilab*. http://www.fnal.gov/.

[27] Yang, Jingqi, Chuanchang Liu, Yanlei Shang, Bo Cheng, Zexiang Mao, Chunhong Liu, Lisha Niu e Junliang Chen: *A cost-aware auto-scaling approach using the workload prediction in service clouds*. Information Systems Frontiers, 16(1):7–18, 2014.

[28] Jonas, Eric, Qifan Pu, Shivaram Venkataraman, Ion Stoica e Benjamin Recht: *Occupy the cloud: Distributed computing for the 99%*. Em *Proceedings of the 2017 Symposium on Cloud Computing*, páginas 445–451. ACM, 2017.

[29] PyWren: *PyWren*. http://pywren.io/.

[30] Dordevic, Borislav S, Slobodan P Jovanovic e Valentina V Timcenko: *Cloud Computing in Amazon and Microsoft Azure platforms: Performance and service comparison*. Em *Telecommunications Forum Telfor (TELFOR), 2014 22nd*, páginas 931–934. IEEE, 2014.

[31] Di Martino, Beniamino, Giuseppina Cretella, Antonio Esposito e Raffaele Giulio Sperandeo: *Semantic representation of cloud services: a case study for Microsoft Windows Azure*. Em *Intelligent Networking and Collaborative Systems (INCoS), 2014 International Conference on*, páginas 647–652. IEEE, 2014.

[32] Carvalho, Alisson Linhares de: *Suporte para execução de máquinas virtuais nativas*. Tese de Mestrado, Universidade Estadual de Campinas, Instituto de Computação, 2015.

[33] Van Dung, Tran e He Jifeng: *A Theory of combinational programs*. Em *Software Engineering Conference, 2001. APSEC 2001. Eighth Asia-Pacific*, páginas 325–328. IEEE, 2001.

[34] Riteau, Pierre, Myunghwa Hwang, Anand Padmanabhan, Yizhao Gao, Yan Liu, Kate Keahey e Shaowen Wang: *A cloud computing approach to on-demand and scalable cybergis analytics*. Em *Proceedings of the 5th ACM workshop on Scientific cloud computing*, páginas 17–24. ACM, 2014.

[35] Zapty: *Forever-Service*. https://github.com/zapty/forever-service.

[36] Ltd., Canonical: *Ubuntu*. https://www.ubuntu.com/.

[37] Laboratório de Mecânica Computacional: *NeoPz*. https://github.com/labmec/neopz.

Apêndice A

Curvas Pareto

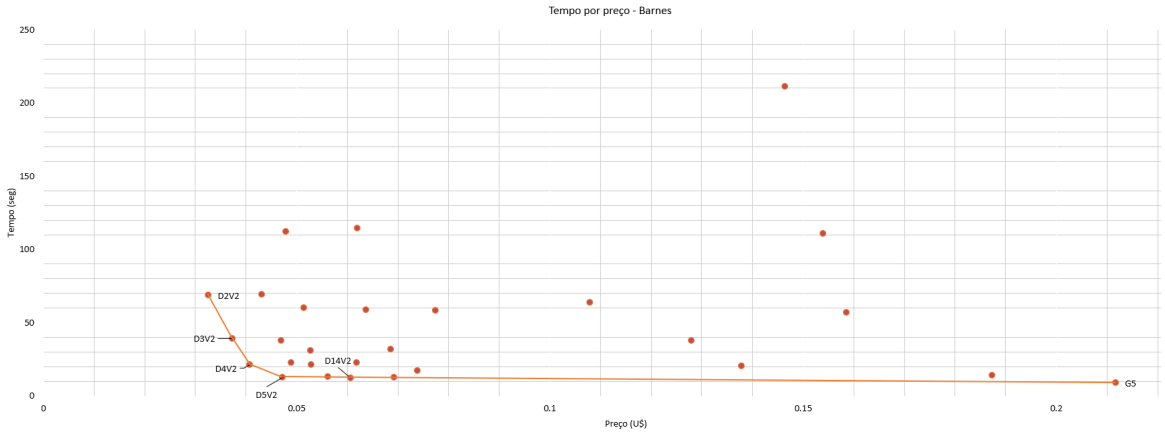

Figura A.1: Curva pareto para *benchmark* barnes.

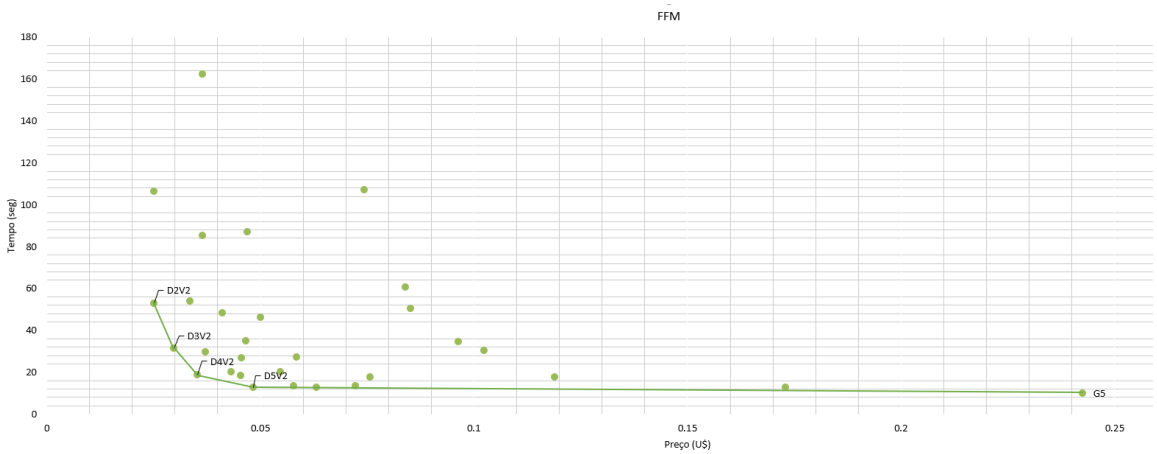

Figura A.2: Curva pareto para *benchmark* ffm.

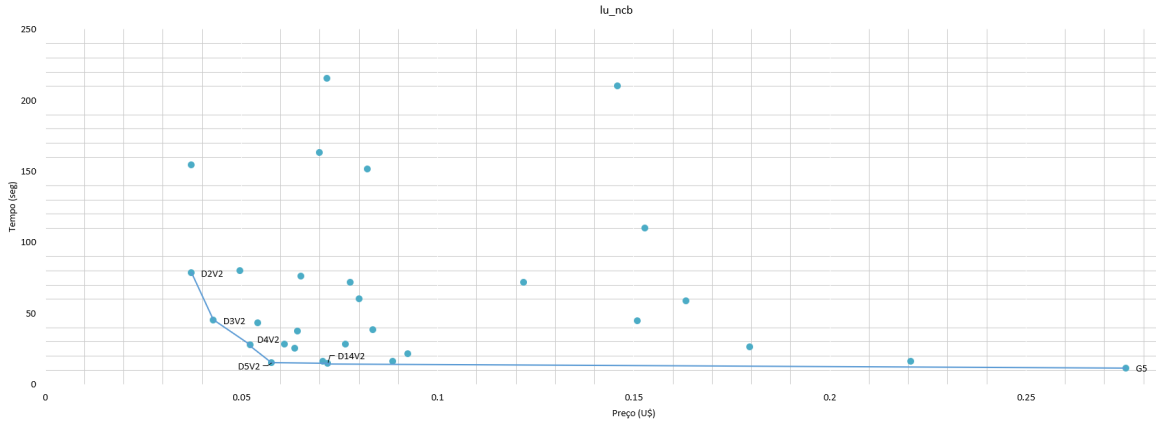

Figura A.3: Curva pareto para *benchmark* lu_ncb.

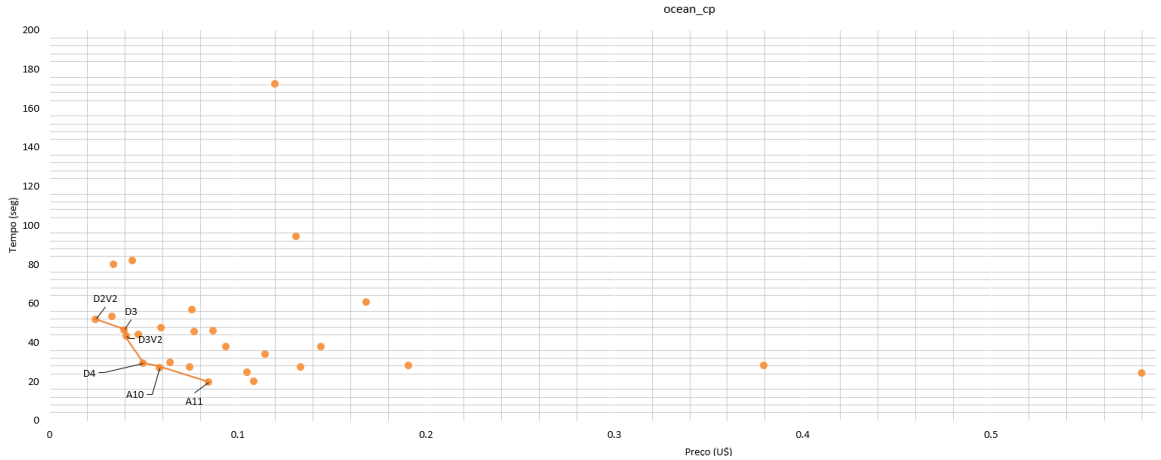

Figura A.4: Curva pareto para *benchmark* ocean_cp.

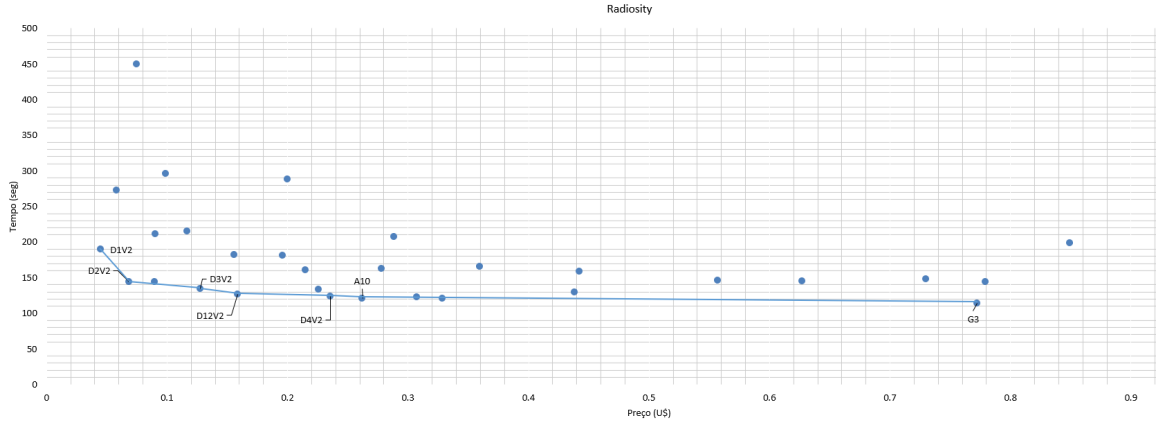

Figura A.5: Curva pareto para *benchmark* radiosity.

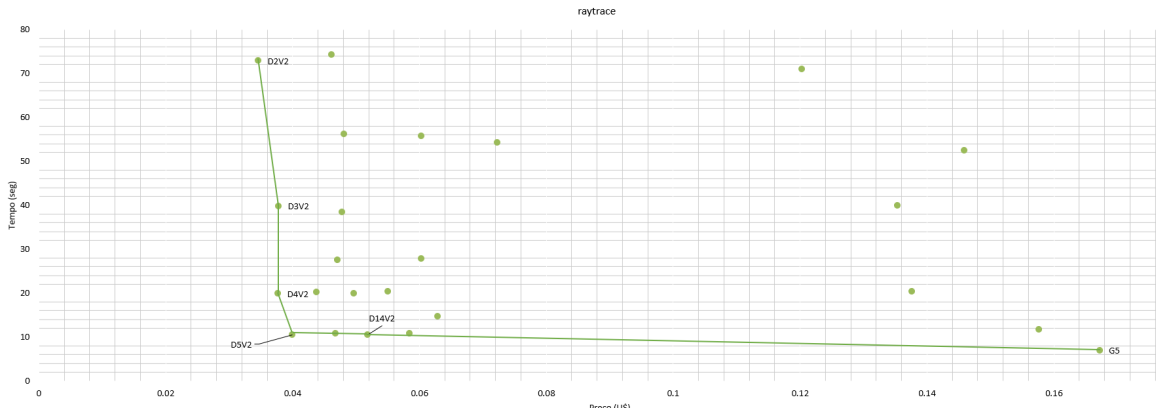

Figura A.6: Curva pareto para *benchmark* raytrace.

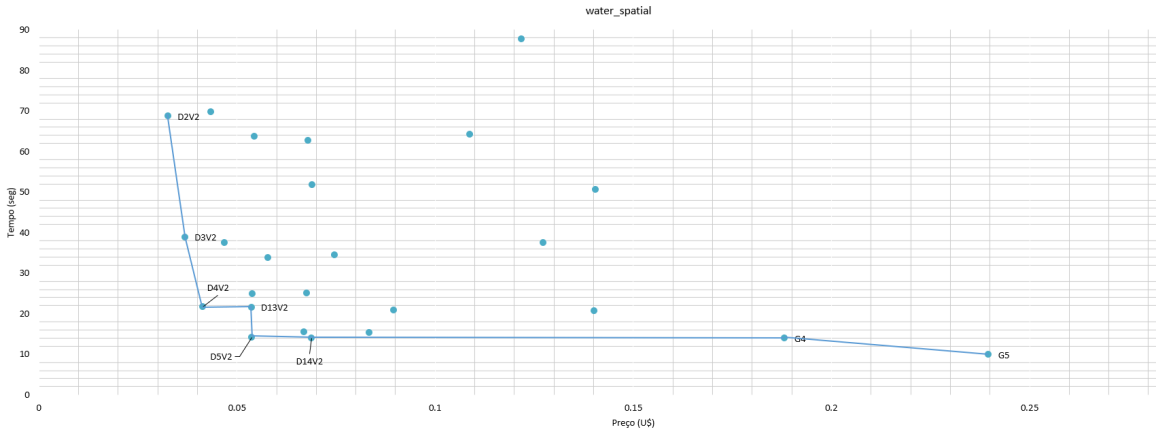

Figura A.7: Curva pareto para *benchmark* water_spatial.