Unioeste - Universidade Estadual do Oeste do Paraná
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
Colegiado de Ciência da Computação
Curso de Bacharelado em Ciência da Computação

Análise comparativa entre algoritmos de escalonamento de aplicações em ambientes de nuvem

Ana Luiza da Rocha Herrmann

CASCAVEL
2018

ANA LUIZA DA ROCHA HERRMANN

ANÁLISE COMPARATIVA ENTRE ALGORITMOS DE ESCALONAMENTO DE APLICAÇÕES EM AMBIENTES DE NUVEM

Monografia apresentada como requisito parcial para obtenção do Título de Bacharel em Ciência da Computação, pela Universidade Estadual do Oeste do Paraná, Campus de Cascavel, aprovada no dia 28/06/2018 pela Comissão formada pelos professores:

Prof. Guilherme Galante (Orientador)
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Marcio Seiji Oyamada
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Luiz Antonio Rodrigues
Colegiado de Ciência da Computação,
UNIOESTE

Cascavel, 31 de julho de 2018

AGRADECIMENTOS

Todos os quatro anos de minha faculdade tive pessoas incríveis lutando junto comigo, e eu dedico este trabalho a todas elas. Primeiramente à minha família que sempre esteve presente, me dando força e compreensão pra conseguir seguir quando eu precisava de um empurrão, e pra me segurar quando percebiam que eu estava indo rápido demais. A atlética Leão Loko, a mais louca do sul do mundo, que me trouxe tantas amizades que seguiram comigo durante esse tempo, e me tiraram da rotina tantas vezes. Aos meus amigos e colegas de sala, que foram poucos, mas que sobreviveram entre todos os quarenta que aqui entraram, só nós sabemos como foi difícil, e o quanto a gente batalhou pra conseguir cumprir todos os prazos (e adiar alguns). Agradeço minha companheira Vitória Benatti, por todas as filosofias, discussões, festas, estudos e por estar do meu lado durante todas as minhas crises de tristeza e de felicidade. Um agradecimento especial a Letícia Torres, que durante esse período foi a pessoa que me acompanhou durante todas as noites mal dormidas, notas baixas, notas altas, trabalhos bem e mal feitos, costelinhas de porco com molho de alho, apostas, risos, lágrimas, e tudo que acompanhou os 4 anos desse curso que só a gente sabe como a gente ama. Agradeço à todos os professores do curso de Ciência da Computação, que passaram para nós o melhor dessa profissão, e me ensinaram muito mais do que eu poderia imaginar, vocês são o exemplo do quanto nós podemos saber. E em especial, agradeço ao meu querido orientador Guilherme Galante, que teve tanta dedicação e paciência durante esse um ano e meio, e me ajudou a perceber que computação não é um bicho de sete cabeças, mas que exige dedicação.

Lista de Figuras

Lista de Tabelas

Sumário

Resumo

Infraestrutura como serviço (IaaS) é um modelo de computação em nuvem, no qual a infraestrutura de servidores, sistemas de rede, armazenamento, e todo o ambiente necessário para o funcionamento são contratados como serviços. Uma forma comum de oferecer esses recursos é por meio de um conjunto de máquinas virtuais (VMs) e serviços complementares. Esse conjunto de VMs pode ser alocado de diferentes formas no datacenter do provedor. Geralmente são transparentes ao usuário e não levam em consideração as características das aplicações que serão executadas. Um problema causado por esse modo de alocação é a heterogeneidade dos valores de latências que podem ser observadas entre pares de instâncias. Assim, o desempenho de aplicações que possuem comunicação via rede, como é o caso de aplicações distribuídas, pode ser afetado significativamente dependendo da forma como suas tarefas são alocadas nas instâncias disponibilizadas. Analisando a literatura técnica, é possível encontrar soluções que visam auxiliar a alocação de aplicações na nuvem. Neste trabalho avalia-se duas soluções, Cloudia e Choreo, com o objetivo de verificar sua efetividade nesta tarefa. As soluções foram avaliadas em uma nuvem IaaS real, a Google Compute Engine, na qual se realizou um conjunto de experimentos com aplicações com diferentes características. De modo geral, ambos os métodos apresentaram melhorias em relação ao escalonamento First Fit, tendo gerado inclusive o mesmo escalonamento para o caso do Cliente-Servidor. Assim, as soluções se mostraram efetivas para a tarefa de escalonar tais aplicações distribuídas.

Palavras-chave: Computação em nuvem, alocação, aplicações distribuídas.

Capítulo 1

Introdução

A computação em nuvem é um modelo de compartilhamento de hardware e software onde grandes *data centers* disponibilizam serviços, infraestrutura e plataformas, como conexão à rede, servidores, armazenamento e aplicações. O modelo provê acesso sob demanda a recursos computacionais configuráveis que podem ser facilmente alocados ou liberados, com mínima necessidade de interação ou gerência por parte do provedor. As principais características da computação em nuvem são o autoatendimento sob demanda; acesso amplo à rede; conjunto de recursos em uma plataforma única; rápida elasticidade; e serviço monitorado [6]. A computação em nuvem utiliza o conceito de pagamento-por-uso, onde o cliente paga somente pelo tempo utilizado e conforme o tipo dos recursos solicitados. Além da rápida escalabilidade e da confiabilidade, a característica que mais atrai clientes é o custo-benefício oferecido, garantido pelo provedor devido ao compartilhamento de recursos e operações em larga escala, que aumenta a utilização da infraestrutura e reduz os custos [7].

Os modelos que definem quais serviços serão disponibilizados são: Software como Serviço (*Software as a Service - SaaS*), onde o cliente pode utilizar as aplicações do provedor que fazem uso da infraestrutura em nuvem; Plataforma como Serviço (*Platform as a Service - PaaS*), em que o provedor disponibiliza uma plataforma onde o consumidor pode implantar e controlar suas aplicações, desde que sejam compatíveis em linguagem, bibliotecas, e serviços suportados pelo provedor, e não mantém controle sobre quaisquer configurações, sistemas operacionais, configurações de rede ou armazenamento; ou Infraestrutura como Serviço (*Infrastructure as a Service - IaaS*), que disponibiliza processamento, serviços de rede, armazenamento e outros recursos computacionais, onde o cliente tem a possibilidade de implantar e executar qualquer software, sistemas operacionais e aplicações que desejar. Por ser um modelo que provê organização física

abstrata, o consumidor não tem controle sobre as configurações da infraestrutura da nuvem, mas tem sobre os sistemas operacionais, armazenamento e aplicações implantadas [6, 8, 9].

De acordo com seu modelo de implantação, nuvens podem ser classificadas como privadas, comunitárias, públicas, ou híbridas. Quando uma organização gerencia e comanda seu próprio *data center*, a nuvem é dita privada. Quando duas ou mais organizações tem interesses em comum, e comandam um *data center* que disponibiliza serviço para tal público, a nuvem é comunitária. Para disponibilização do serviço em nuvem ao público geral, ou seja, nuvem pública, uma organização governamental, empresa ou universidade faz a gerência da infraestrutura. No caso da nuvem se encaixar em duas ou mais classificações anteriormente apresentadas, é dita híbrida [6].

No modelo IaaS, foco deste trabalho, a disponibilização dos serviços pode ser feita diretamente nas máquinas físicas, ou, como será estudado neste trabalho, utilizando virtualização, termo utilizado para referenciar a abstração dos recursos de um computador. Quando utilizamos sistemas virtualizados, a máquina física hospedeira está dividida entre máquinas virtuais (*Virtual Machines - VM*), e um gerenciador destas máquinas e dos recursos do *host*, chamado de *Virtual Machine Monitor* (VMM), ou *hypervisor*, que é responsável pela camada de software que comunica as diferentes plataformas que estão sendo executadas [10]. Cada VM é uma cópia eficiente e isolada de uma máquina real [11].

A alocação das máquinas virtuais na infraestrutura do provedor depende de um SLA (*Service Level Agreement*), que dita condições de recursos, isolamento, e disponibilidade. Para satisfazer as exigências das inúmeras aplicações, os servidores em nuvem oferecem serviços com diferentes preços e configurações de recursos. Os tipos de instância oferecem diversas combinações de quantidade de núcleos de processamento, memória, largura de banda, entre outros requisitos de configuração das máquinas [12, 7].

Para a alocação das VMs, os provedores utilizam apenas informações de seu próprio *data center*, fazendo com que os componentes sejam alocados de maneira agnóstica em relação às aplicações que serão executadas. Assim, tais instâncias podem ser alocadas em máquinas físicas muito próximas ou muito distantes, em diferentes data centers, por exemplo. Esta estratégia de alocação pode conduzir à heterogeneidade de latência de rede entre as instâncias. Além disso, a virtualização e o uso compartilhado dos recursos de rede entre as aplicações resulta em *jitter*.

O estudo de Tao Zou [7] aborda a diferença das latências e o *jitter* na nuvem, debatendo sobre esta heterogeneidade com relação ao tempo de comunicação nas aplicações distribuídas, e como este tempo pode trazer aumentos significativos ao tempo de resposta das aplicações. A diferença significativa do tempo de resposta das aplicações pode ser resultado também da diferença das características de hardware das máquinas físicas, além da diferença na largura de banda.

As diferenças de conexão entre as VMs disponibilizadas ao cliente, que fazem com que alguns pares de instâncias estejam conectados com menor tempo de latência do que outros, não são informados em nenhuma API do provedor, e as informações sobre a topologia de comunicação entre as máquinas virtuais também não são fornecidas [3].

Por falta de informação, as aplicações em nuvem, que tratam em sua grande maioria de aplicações distribuídas ou paralelas e intensivas em uso de rede, são afetadas negativamente pela alocação independente das características da aplicação. Tais aplicações, que podem operar entre dezenas de máquinas, têm suas componentes coordenadas através da rede e realizam troca de grandes quantidades de dados, de forma que o aumento da latência de comunicação e tempo de resposta são de grande influência no tempo total de execução [7, 13].

Analisando a literatura técnica na área, é possível encontrar diferentes abordagens com o intuito de melhorar a alocação de aplicações distribuídas na nuvem. As propostas têm como objetivo coletar informações da rede e utilizá-las como parâmetros para a realização da alocação das aplicações. De modo geral, duas abordagens distintas podem ser usadas. Na primeira, o desenvolvedor executa as medições necessárias, otimiza a alocação dentre as VMs já disponibilizadas pelo provedor e realiza a alocação da aplicação conforme o necessário [3, 13]. Já na segunda abordagem, há participação do provedor da nuvem, com o intuito de melhorar a escolha das VMs que serão disponibilizadas, ou como maneira de obter, de forma indireta, informações que auxiliam nas melhores escolhas [4, 5].

Nesse contexto, este trabalho tem dois objetivos: (1) testar a influência da latência em aplicações distribuídas e (2) comparar os métodos de escalonamento de aplicações disponíveis na literatura técnica. Mais especificamente, avalia-se as soluções Cloudia [3] e Choreo [13], que se enquadram na classe dos métodos que não utilizam informações ou mecanismos do provedor de nuvem para melhorar a alocação das aplicações.

As soluções foram avaliadas em uma nuvem IaaS real, a Google Compute Cloud, na qual se

realizou um conjunto de experimentos com aplicações com diferentes características. De modo geral, foi possível notar que o escalonamento correto das aplicações distribuídas tem grande influência no tempo de execução das aplicações. ClouDiA e Choreo apresentaram soluções efetivas para melhorar a tarefa de escalonar aplicações distribuídas nos modelos Cliente-Servidor e Todos para Todos distribuídas.

O restante do texto está organizado da seguinte forma. A Seção 2 descreve o problema da alocação, dos tempos de comunicação, e os métodos para o escalonamento de aplicações encontradas na literatura. A Seção 3 descreve as metodologias de implementação dos métodos de escalonamento. A Seção 4 descreve os resultados obtidos. E a Seção 5 descreve a conclusão perante aos resultados.

Capítulo 2

Escalonamento de Aplicações em Nuvens Públicas

2.1 Contextualização do Problema

A computação em nuvem tem se tornado uma opção de computação muito bem sucedida, provendo serviços e recursos sob demanda, sem a preocupação de localização dos dados e servidores. A infraestrutura de uma nuvem consiste em data centers, que são monitorados e mantidos por grandes provedores. Os serviços podem ser contratados através da internet, e atraem clientes pela simplicidade, e pelo custo-benefício que o serviço provê [14].

Para atender o rápido crescimento da computação, do armazenamento e da comunicação, os provedores da nuvem realizam a alocação de recursos em larga escala. Conforme a quantidade de recursos é ampliada, o aumento do uso da computação em nuvem faz com que a comunicação entre os componentes também aumente, assim, o tráfego de dados entre as máquinas dentro dos *data centers* dos provedores tende a aumentar. A Figura 2.1 ilustra um estudo da Cisco Systems Inc. [1] sobre o crescimento do tráfego de dados mundial, enfatizando a projeção de que em 2019 83% de todo o tráfego de dados será gerado por servidores em nuvem, que crescerá de 2 Zettabytes por ano, para mais de 8 Zettabytes por ano, enquanto a quantidade de dados dentro de *data centers* tradicionais se mantém a mesma. Além disso, o estudo também relata que a maior parte do fluxo de dados é gerado pela comunicação *intracloud*, número que estará próximo a 73.1% no ano de 2019 [15].

O aumento do fluxo de dados dentro dos data centers é gerado principalmente pela comunicação dos componentes das aplicações alocadas em nuvem, que são em sua grande maioria paralelas ou distribuídas. Tais aplicações são altamente afetadas pelo tempo de comunicação

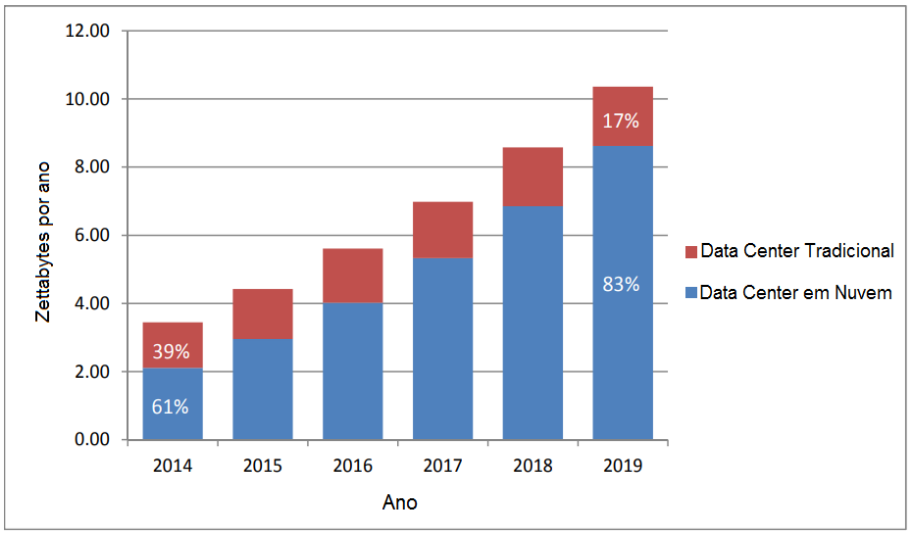

Figura 2.1: Crescimento do tráfego mundial de dados dentro de data centers. Fonte: adaptado de [1]

entre os componentes, e normalmente não tem recursos de rede garantidos. Nesses casos, a largura de banda é dividida entre inúmeros clientes, e consequentemente a velocidade de conexão que será atingida depende da quantidade de tráfego nos links que estão sendo utilizados, e da quantidade de VMs que estão alocadas nas máquinas físicas [13, 15, 16]. Além disso, Greenberg *et al.* [17] comentam sobre as dificuldades de comunicação dentro dos *data centers*, devido ao não preparo das arquiteturas de rede para disponibilizar largura de banda suficiente dentro destas entidades, o que pode gerar congestionamento e *hot spots* de utilização dos links, enquanto outras áreas estão com a capacidade de conexão ociosas.

A dificuldade de lidar com os problemas de comunicação dentro da nuvem vem também da liberdade que o provedor necessita para conseguir balancear o uso da infraestrutura. Com a virtualização das máquinas físicas, e a semelhança aparente entre as VMs que são fornecidas ao cliente, as n máquinas virtuais contratadas pelo cliente podem ter diferentes níveis de distância, já que mesmo com a escolha da zona (*data center*) das VMs do cliente, não tem como saber se elas estarão na mesma máquina física ou no mesmo rack. Essa informação sobre onde a VM do cliente está não é fornecida exatamente porque o provedor precisa de liberdade para alocar e realocar as máquinas onde é mais conveniente para o uso da infraestrutura. Para o cliente a falta destas informações dificulta a otimização de aplicações que realizam grande troca de dados e utilização da rede, e torna impossível explorar possíveis gargalos [18].

	Requerimentos da Infraestrutura da Nuvem		
Aplicações	Intensidade de Computação	Largura de Banda	Latência de Rede*
Teste & Desenvolvimento	Alta	Baixa	Alta
Navegação na Web	Baixa	Baixa	Alta
Backup & Recuperação	Baixa	Alta	Alta
Email & Calendário	Baixa	Baixa	Alta
HRM	Média	Baixa	Alta
Gestão de Documentos	Baixa	Alta	Média
CRM	Média	Baixa	Média

**RTD (Round Trip Time)*

Tabela 2.1: A influência da latência em aplicações

Como a alocação das VMs sofre influência de inúmeros fatores, as máquinas disponibilizadas para o cliente podem ter diferentes características. Mesmo com a solicitação de máquinas homogêneas, elas estarão sujeitas a diferentes pontos de acesso, influência do uso das outras VMs que estão na mesma máquina física, além de que, para alguns provedores, existe um limite de alocação de VMs por zona (*data center*), assim, as máquinas terão diferenças de tempo de comunicação ainda mais expressivos entre si.

Todas essas variações fazem com que alguns pares de instâncias tenham menores tempos de latência e largura de banda mais ociosa do que outros, e esta diferença afeta o tempo de comunicação e deteriora o serviço das aplicações, que como já citado por Lacurts *et al.* [13] são em sua grande maioria intensivas em uso de rede e sensíveis a latência [3].

Como mostra a Tabela 2.1, a latência de rede se tornou, para inúmeras aplicações, de grande importância, tendo mais relevância na qualidade do serviço do que a largura de banda em alguns casos. Os provedores Google [19] e Amazon [20] notaram grandes perdas de vendas e tráfego quando páginas demoram para carregar. Meio segundo de atraso pode causar até 20% de queda no tráfego da Google, e um décimo de segundo de atraso pode causar queda de 1% nas vendas da Amazon.

O gráfico da Figura 2.2 ilustra a importância da latência de rede pela taxa de perda de visualizações e conversões, que quando o tempo de latência atinge os 10 segundos é de 42% e 46%, respectivamente; e pela taxa de rejeição do site, que chega a 135% no mesmo tempo [2].

Em grande parte das aplicações, a sensibilidade à latência de comunicação se aplica a um conjunto de componentes, e não necessariamente a todas as VMs que serão utilizadas pelo cliente. Toma-se como exemplo uma aplicação *e-commerce*, que normalmente é dividida em

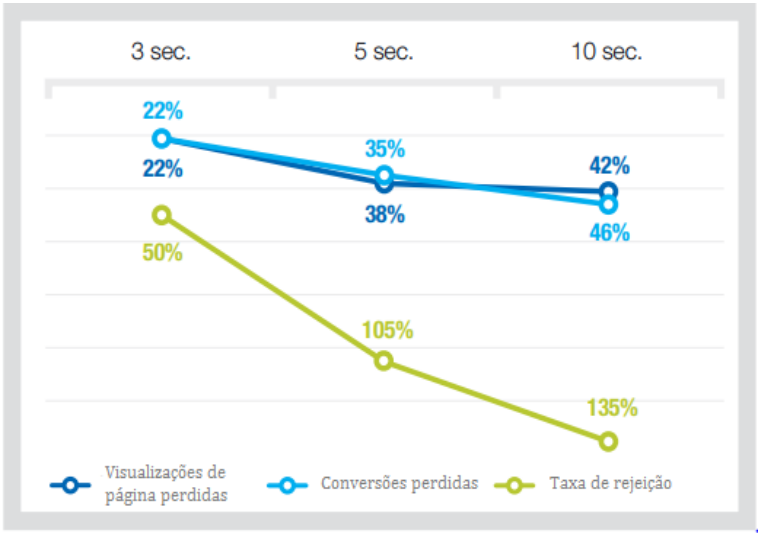

Figura 2.2: As consequências do atraso no tempo de carregamento de sites. Fonte: adaptado de [2]

3 camadas: de apresentação (módulo web); camada lógica (servidor da aplicação); e camada de dados (banco de dados) [21]. Cada um dos componentes de computação tem demandas de comunicação entre si, e entre seus blocos de dados. Como consequência, o tempo total de execução depende do tempo de comunicação entre os componentes de processamento e de acesso à dados. A solução utilizada pelos provedores é realizar o escalonamento da aplicação ignorando as diferenças dos componentes e suas necessidades [15]. Porém, as propostas estudadas neste trabalho defendem que melhorando tal alocação, é possível melhorar a qualidade do serviço e diminuir o tempo de execução das aplicações .

LaCurts *et al* [5] e Agache *et al* [4], propõem APIs de integração entre o cliente e o provedor, para a apresentação de requisitos e padrões das aplicações, de modo que o provedor tenha a capacidade de realizar melhorias na alocação, e na comunicação dos componentes mesmo depois de alocados. As duas abordagens utilizam ferramentas que precisam ser executadas pelo provedor da nuvem, e como nas plataformas de nuvem pública disponíveis hoje não permitem esse tipo de API, Tao Zou *et al* [3] e LaCurts *et al* [13] apresentam uma abordagem que tem como ponto de vista somente o cliente, e pode ser aplicada imediatamente pelos desenvolvedores.

2.1.1 Aplicações Distribuídas

As aplicações em rede alocadas em nuvem são compostas por processos que se comunicam entre si por meio de uma rede. No contexto de comunicação de processos, podemos ter o cliente, que é aquele que inicia a comunicação, e o servidor, que só se comunica após solicitação, e fica sempre em funcionamento aguardando comunicações. As aplicações Web são um exemplo de cliente-servidor, já que um servidor Web está sempre em funcionamento aguardando requisições, e o cliente é aquele que inicia somente quando é necessário. Nesta arquitetura, não há comunicação entre os clientes, somente entre os clientes e o servidor. Além da Web, o *File Transfer Protocol*(FTP), e o e-mail também utilizam a arquitetura cliente-servidor em suas aplicações [22].

Algumas arquiteturas não fazem uso de uma máquina central (o servidor), e todos os processos podem se comunicar entre si, é o exemplo da arquitetura par a par. O nome par a par (mais conhecido como *peer-to-peer* - P2P) vem de que as comunicações acontecem em pares, utilizando a comunicação direta entre duplas de máquinas. Essa arquitetura é muito utilizada em aplicações que possuem alta quantidade de tráfego de dados, por exemplo em aplicações de compartilhamento de arquivos como o BitTorrent, ou a telefonia por Internet como o Skype [22].

Outra arquitetura que não faz uso de uma máquina central é a comunicação todos-para-todos, onde todos os processos podem se comunicar entre si. Esta comunicação é utilizada em aplicações paralelas (MPI), e protocolos de broadcast, que são aplicações que podem utilizar transferência de arquivos dos mais variados tamanhos.

Todas as aplicações, por mais que não funcionem como o Cliente-Servidor, sempre terão um processo cliente e um processo servidor em uma comunicação. Isso quer dizer, que em aplicações como Peer-to-Peer ou todos-para-todos, por mais que não exista um servidor central, todas as comunicações entre processos são iniciadas por um processo cliente, e recebidas por um processo servidor [22], essa lógica foi utilizada neste trabalho.

2.2 Estado da Arte

Nesta seção são apresentados o funcionamento e principais resultados das propostas disponíveis na literatura.

2.2.1 ClouDiA

No trabalho de Tao Zou *et al* [3], apresenta-se o ClouDiA (*Cloud Deployment Advisor*), que tem como objetivo verificar como os desenvolvedores podem melhorar a alocação de aplicações distribuídas dentro da nuvem pública. O ClouDiA toma como base a visão do desenvolvedor, que poderá realizar as mudanças e otimizar as aplicações imediatamente, sem necessitar mudanças nos servidores da nuvem.

Duas observações são importantes: (i) se a alocação dos componentes da aplicação for feita de forma compatível com as necessidades, más conexões podem ser prevenidas; (ii) se forem alocadas instâncias a mais, aumentamos as chances de melhores conexões serem encontradas.

O ClouDiA classifica as aplicações sensíveis à latência em dois grupos: computação de alta performance, onde o fator principal é o tempo de processamento, que é afetado pela latência do pior link de comunicação; e serviços interativos, que apresentam tempo de resposta como fator de maior importância, e podem ser altamente afetados pela latência nos caminhos críticos.

O funcionamento do ClouDiA admite que a aplicação estará com conexão estável à rede, é ilustrada na Figura 2.3 e segue os passos:

1. Alocação das instâncias

   O contratante especifica o grafo de comunicação da aplicação, juntamente com o número máximo de instâncias que serão alocadas. O ClouDiA alocará um número maior de máquinas para aumentar as chances de encontrar uma implantação com menor latência de comunicação.

2. Obter Medições

   O desafio deste passo é obter as latências entre os pares de máquinas rapidamente, de forma que o tempo de cálculo não afete a aplicação.

3. Pesquisar Implantações

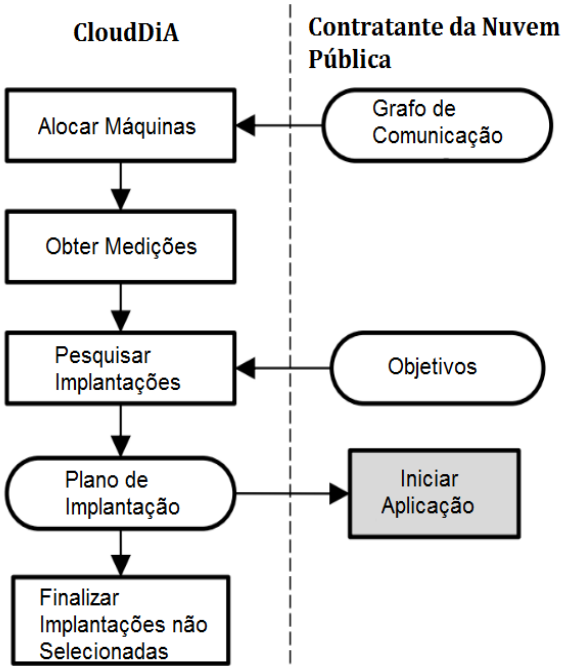

Figura 2.3: Arquitetura do ClouDiA. Fonte: adaptado de [3]

Utilizando as medições obtidas, o grafo de comunicação e os objetivos de otimização especificados pelo contratante, o ClouDiA pesquisa pelo melhor plano de implantação da aplicação, de forma que evite os piores links de comunicação. Como o ClouDiA trabalha com dois tipos de latência (HPC e serviços interativos), são formuladas duas variações do problema da implantação.

4. Selecionar o Plano de Implantação e Iniciar a Aplicação

5. Finalizar Implantações não Selecionadas

   Após definido o plano de implantação, todas as máquinas que não foram selecionadas são finalizadas.

Para calcular as latências, o ClouDiA propôs três possibilidades: passagem de token; não coordenado, que tem como objetivo aumentar a escalabilidade e utilizar envios paralelos de pacotes de dados; e coordenado, onde uma instância extra é adicionada para coordenar o processo de medição, e escolher os destinos dos pacotes de dados. Os autores não conseguiram executar nenhum dos métodos em um tempo que não afetasse significativamente o tempo final dos experimentos. Os autores executaram tentativas para obter distância de rede entre os nodos,

utilizando o IP para calcular a distância dentro da arquitetura, e o contador de saltos, o retorno destas métricas não é a latência mas foi possível realizar uma aproximação. Para outros dois cenários testados separadamente, foi utilizado o envio de pacotes TCP de 1KB para retorno do *Round Trip Time*, a captura da heterogeneidade da latência foi menor nesta métrica.

Eficiência do ClouDiA:

- *Setup*

  No artigo original, para a avaliação do sistema, foram utilizadas de 100 a 150 instâncias do Amazon EC2 [23]. A quantidade de máquinas alocadas foi 10% maior do que o necessário no primeiro teste, havendo variação no segundo teste.

- Eficiência

  ClouDiA alcançou de 15% até 55% no tempo de execução ou tempo de resposta entre 5 alocações para 3 aplicações em relação à não aplicação do método.

- Alocação de instâncias a mais

  O ClouDiA atingiu 38% na melhoria da performance quando utilizou 50% de instâncias extras. Sem a alocação de nodos a mais, foi possível atingir 16% de melhora, e com 10% a mais de nodos, a melhoria foi de 28%. É importante ressaltar que apesar do ClouDiA finalizar as instâncias, ainda são cobradas as taxas de uso de pelo menos uma hora, gerando mais custos, por isso a importância de encontrar um ponto de equilíbrio entre a quantidade de máquinas alocadas.

2.2.2 Choreo

Proposto por LaCurts *et al*, Choreo [13] tem o contexto geral de que um cliente fez a solicitação de uma certa quantidade de máquinas ao provedor. Após receber acesso às VMs, o cliente tem uma pergunta: como alocar uma ou mais aplicações, para atingir a melhor desempenho possível? Normalmente, o desempenho se refere ao tempo de execução de uma aplicação.

Em Choreo [13] são feitas duas principais contribuições: a primeira é um sistema para alocação *network-aware* de aplicações, chamado Choreo, que clientes da nuvem pública podem utilizar para alocar um conjunto de aplicações na infraestrutura da nuvem; a segunda contribuição é a comparação e avaliação do desempenho do Choreo em relação com outros métodos

de alocação que não utilizam taxas de rede ou padrões de comunicação entre tarefas. Neste trabalho, será estudada somente a primeira contribuição.

O Choreo tem três sub-sistemas: (1) identificar as tarefas da aplicação do cliente; (2) medir o desempenho da rede entre as VMs disponibilizadas; e (3) alocar e monitorar as tarefas. O método funciona tentando alocar os pares de tarefas que realizam maior transferência de dados nas VMs que tem conexão mais rápida.

1. Identificação da Aplicação

   Choreo utiliza ferramentas de monitoramento da rede para identificar os padrões de comunicação da aplicação. A saída desta etapa é uma matriz *A* onde as linhas e colunas são as tarefas da aplicação. Cada entrada A_{ij} corresponde à quantidade de bytes enviados da tarefa *i* para a tarefa *j*.

2. Obter Medições

   Obtém as informações da rede entre cada par de VM coletando a quantidade de bytes enviados entre as tarefas, para então identificar o *throughput* do TCP entre as máquinas. A medição deve acontecer de forma rápida, para que as aplicações sejam alocadas em tempo conveniente e não haja mudança da rede entre as medições e a alocação das tarefas.

3. Alocação da Aplicação

   Depois de identificar as comunicações e obter as medições, é realizada a alocação das tarefas da aplicação. O problema da alocação é formulado como um ILP (*Integer Linear Program*) e para diminuir o tempo de resolução é resolvido com um algoritmo guloso.

4. Lidando com Múltiplas Aplicações

   O Choreo refaz as medições toda vez que uma nova aplicação do contratante é iniciada. Para obter melhores resultados, a cada T minutos as medições são refeitas e caso seja necessário é possível realizar a migração das tarefas. Quanto menos custosa é a migração, menor é T.

Os autores avaliam o uso do Choreo em dois cenários. O primeiro cenário simula clientes que querem executar múltiplas aplicações simultaneamente, e todas as informações necessárias

são conhecidas. Nesse caso, Choreo mostrou melhorias de 8% a 14% no tempo de execução de 70% das aplicações que estavam sendo alocadas. O segundo cenário, as aplicações não tem todas as variáveis conhecidas a priori, e vão sendo alocadas conforme chegam as informações, portanto, as aplicações são alocadas uma a uma. Neste caso, Choreo atingiu uma melhoria de 22% a 43% no tempo de execução de aproximadamente 90% das aplicações em relação aos escalonamentos utilizando *round-robin* e escalonamento aleatório.

2.2.3 CloudTalk

CloudTalk [4], proposto por Agache *et al*, levanta as dificuldades de realizar otimizações em nuvens públicas. Pela falta de APIs que realizem a comunicação entre clientes e servidores, os clientes não sabem qual é a infraestrutura da rede e topologia e *status* de armazenamento, e os provedores não têm conhecimento sobre as aplicações do cliente, tornando impossível otimizar os processos de escalonamento e alocação das aplicações.

O CloudTalk propõe uma API que possibilita otimização da aplicação e do uso da infraestrutura, enquanto preserva a flexibilidade de alocação de VMs pelo provedor. O objetivo é prover uma otimização de aplicações distribuídas e do uso da rede em nuvens públicas. Enquanto os clientes podem otimizar as tarefas sem informações sobre a rede, provedores podem manter as conexões de rede sem fornecer informações sobre a infraestrutura, e sem controlar as aplicações dos clientes.

A API disponibilizada é utilizada pelos clientes para expressar padrões de tráfego de dados que eles desejam executar, eles utilizam a linguagem CloudTalk para descrever cenários que podem acontecer, e as opções de fluxo. Enquanto o provedor da nuvem executa um servidor CloudTalk em cada gerenciador de máquina virtual (VMM), que utiliza as informações do tráfego atual e da topologia para responder as solicitações das aplicações, o objetivo é que o provedor dê a sugestão da melhor alternativa para cada solicitação.

Na Figura 2.4, a VM1 faz a requisição para o servidor de qual máquina deveria solicitar a leitura, e o servidor recolhe informações dos servidores de *status* das duas opções propostas pela aplicação, e responde com a melhor alternativa. Se um dos servidores de status não responderem em um tempo pré-determinado, é tomado como servidor sob grande carga.

Para estimar o tempo de finalização de uma tarefa, o CloudTalk fornece duas possibilidades:

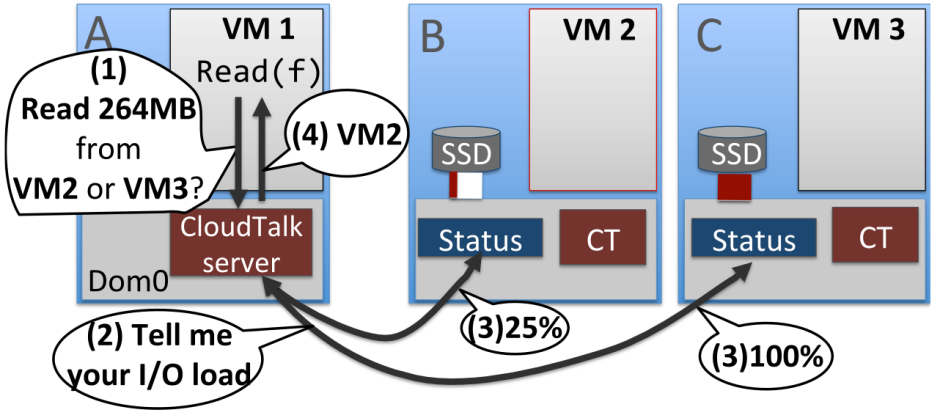

Figura 2.4: Arquitetura do CloudTalk: aplicação do cliente faz o pedido sobre qual é a melhor maneira de executar a tarefa. Fonte: adaptado de [4]

um simulador a nível de pacote, ou a nível de fluxo. A primeira alternativa é mais precisa, e consegue capturar os efeitos a nível de pacote, como efeitos de um *broadcast*, porém, pode ser muito lenta quando a quantidade de pacotes é muito grande. Já o simulador a nível de fluxo determina aritmeticamente uma taxa para cada fluxo de dados, e calcula as taxas iterativamente até que esteja estabilizada. Para grandes transferências de dados a segunda alternativa é mais precisa, porém não funciona corretamente para fluxos pequenos. Quando a tarefa do cliente envia a solicitação, envia também qual dos dois métodos deseja utilizar.

A avaliação do CloudTalk foi executada com três classes de aplicação que podem ser beneficiadas: Computação de MapReduce (Hadoop), um sistema de arquivos distribuído (HDFS) e pesquisa Web (Solr). As aplicações foram alteradas para gerar os pedidos e executar as respostas, quando fosse possível. A performance do CloudTalk foi examinada pela implementação do servidor e sua escalabilidade, e também comparando a performance do Hadoop, HDFS e Solr com e sem as otimizações. Os resultados mostraram melhorias de 15% a 100% na performance das aplicações.

2.2.4 *A Plan for Optimizing Network-Intensive Cloud Applications*

Este método foi proposto por LaCurts *et al* [5], e defende que para aplicações intensivas em uso de rede, a abordagem necessária é que as aplicações apresentem de forma concisa um conjunto de atributos da aplicação, que serão considerados pelo provedor na escolha das VMs.

E depois de alocadas as máquinas virtuais, o cliente deveria ter a possibilidade de solicitar um serviço para melhor escalonamento da aplicação entre as VMs.

Tal arquitetura poderia melhorar o desempenho, e balancear as necessidades do cliente e do provedor, como: minimizar tempo de conclusão da tarefa, minimizar custo monetário, maximizar o número total de aplicações dado um orçamento, minimizar a variação do tempo de conclusão das tarefas, tolerância a falhas e elasticidade para o cliente; e requerimentos de migração e isolamento do provedor.

Funcionamento proposto:

1. Apresentação de demandas e objetivos

   As aplicações podem expressar qualquer um ou todos os requisitos: rede; número de bytes que serão transferidos; largura de banda; núcleos de processamento e porcentagem necessária dos núcleos; requerimentos de tolerância à falhas; conjunto de máquinas que precisam estar em diferentes redes, e quão separadas as VMs precisam estar (por exemplo, diferentes *data centers*); demanda de elasticidade e a função de aumento de escala (por exemplo, ative uma nova máquina toda vez que a média da latência de rede é maior que 200ms). A granularidade das demandas neste trabalho são as tarefas: grupos de conexões e processos que não podem ser divididos em mais de uma máquina.

2. Medições

   Para realizar as etapas de otimização e alocação, são necessárias informações sobre o estado atual da rede. Além de perguntas como: *qual é a quantidade de dados que irão trafegar na rede?* e *quão frequentemente essa quantidade irá mudar?* Também é importante saber a topologia da rede.

3. Otimização

   Os problemas de otimização são resolvidos em programas como o CPLEX [24].

4. Alocação

   Após encontrar o melhor conjunto de máquinas, as tarefas são distribuídas entre as VMs recomendadas. Para isso, o provedor da nuvem exporta uma API que irá especificar

a classe da tarefa, tal que os clientes podem incorporar às aplicações. Além disso, provedores podem precisar prover suporte para usar *tags* nas redes que pertencem à mesma aplicação, essa informação será utilizada para realizar melhor planejamento em nível de rede.

Na Figura 2.5 cada um dos retângulos são parte do funcionamento proposto. As flechas demonstram o fluxo da informação entre as partes da arquitetura, da aplicação e da rede na nuvem. Na direita do pontilhado são as partes que serão do provedor da infraestrutura.

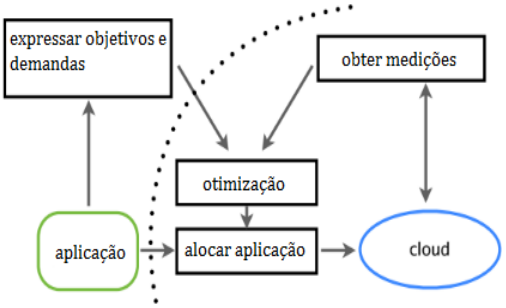

Figura 2.5: Funcionamento do método. Fonte: adaptado de [5]

Para avaliar o método proposto, foram executados testes preliminares na plataforma Amazon EC2 [23]. Nestes experimentos, foram medidos os *throughputs* dos links na topologia do EC2, e determinados alocações ótimas para minimizar o tempo de execução para determinadas aplicações.

Para os testes, foram utilizadas aplicações distribuídas com topologias de comunicação mais comuns, como: pares aleatórios, um para muitos, todos para todos, e todos para todos com múltiplos usuários. Para fins de comparação, tais aplicações foram alocadas seguindo as estratégias: aleatório, porcentagem máxima e mínima de uso de CPU nas máquinas, e quantidade mínima de máquinas utilizadas. As melhorias foram de 29.3% em relação ao escalonamento aleatório, 31.1% em relação ao escalonamento de porcentagem máxima e mínima de uso de CPU, e 19.4% em relação ao escalonamento que de quantidade mínima de máquinas utilizadas.

2.3 Considerações Finais

Dentro das quatro propostas expostas neste trabalho, é possível separá-las em duas abordagens: aquelas que necessitam de interação e de informações fornecidas pelo provedor (CloudTalk [4], Lacurts *et al.* [5]); e aquelas que são aplicáveis sem nenhum tipo de alteração nas infraestruturas, ou necessitam de informações do provedor que hoje não são fornecidas (Cloudia [25], Choreo [13]).

Todas as propostas apresentaram melhoria no tempo de execução das aplicações, porém, levando em consideração que as propostas aplicáveis hoje são aquelas que não dependem de interação com o provedor, as duas propostas que fazem uso de informações que podem ser obtidas em nuvens públicas serão comparadas neste trabalho.

Capítulo 3

Comparando os Algoritmos de Escalonamento

Em toda aplicação distribuída existem inúmeras tarefas que serão executadas simultaneamente, e tais tarefas são escalonadas dentro das VMs solicitadas e disponibilizadas pelo provedor. Neste trabalho, é importante enfatizar que cada parte da aplicação será chamada de tarefa, enquanto cada máquina virtual poderá ser chamada de máquina ou VM.

3.1 Captura de Latência entre as VMs

Para realizar a medição da latência entre as VMs contratadas, o método utilizado neste trabalho foi o envio de pacotes *ping* entre cada par de VM. O *ping* faz uso do envio de uma sequencia de ICMP ECHO_REQUEST entre duas máquinas, e utiliza o RTT (*Round Trip Time*) para calcular a latência. A quantidade de pacotes enviados varia conforme o objetivo da medição, no caso deste trabalho, foram enviados 30 pacotes de 64 bytes, tamanho próximo do tamanho máximo de envio. Esse método foi utilizado para tentar captar também a variação dos tempos no envio da mesma solicitação.

3.2 Implementação

Para a implementação das abordagens de escalonamento de aplicações (Choreo[13] e ClouDiA[25]), foi realizada a padronização de algumas funções utilizadas por ambos os métodos. Esta padronização tem como objetivo facilitar a comparação entre as abordagens, e inclui funções de medição de tempo, tratamento de arquivos e saída de resultado.

	1	2	3	4
1	0	132990	71820	127300
2	189060	0	52890	214820
3	56380	230290	0	199080
4	78500	20440	187710	0

Para o funcionamento dos métodos de escalonamento, são necessários dois arquivos: o grafo de custo de comunicação, que informa os tempos de comunicação entre as VMs; e o grafo que informa a quantidade de dados (em quilobytes) que serão enviados entre as tarefas da aplicação.

O arquivo do grafo de custo de comunicação possui uma matriz triangular superior que informa o tempo de comunicação entre cada par de VM. O tempo de comunicação é a média de uma sequência de tempos obtidos pelo *ping*, conforme descrito na Seção 3.1.

O arquivo do grafo de comunicação da aplicação contém uma matriz com a quantidade de linhas e colunas igual ao número de tarefas da aplicação. A matriz indica a quantidade de bytes do arquivo que é enviado da tarefa *i* para a tarefa *j*. Assim, se o valor de [*i, j*] é igual a 204800, quer dizer que a tarefa *i* envia um arquivo de 200kb para *j*. Os valores foram distribuídos de maneira aleatória conforme a topologia das aplicações que foram testadas. Um exemplo é mostrado na Tabela 3.2, neste exemplo, a tarefa 1 irá enviar 132990 bytes para a tarefa 2; 71820 bytes para a tarefa 3; e 123300 bytes para a tarefa 4; e assim por diante.

As duas abordagens foram implementadas utilizando os mesmos algoritmos de leitura de arquivo e alocação de matrizes. A prioridade de alocação também é igual, ambas utilizam a aresta do grafo de comunicação com maior quantidade de bytes trafegados. Essa semelhança também faz com que a ordenação dos vetores de comunicação e tempos sejam iguais. Assim, a única diferença no tempo de execução das abordagens é a lógica utilizada para realizar o escalonamento das aplicações.

A priorização da aresta com envio do maior arquivo entre duas tarefas é devido ao impacto que a latência de comunicação tem na duração da execução. A lógica utilizada é que se o arquivo é maior, a quantidade de pacotes enviados entre as VMs é maior, e a soma da latência dos envios se torna maior. Esta situação não se aplica à todos os casos, mas é utilizada por ser compatível com grande parte das ocorrências.

A saída dos métodos é um arquivo com os IPs das máquinas que terão cada uma das tarefas, então se a tarefa 1 ficar na máquina 10, na primeira linha do arquivo estará gravado o endereço

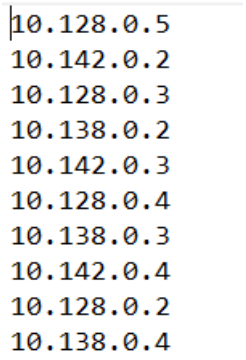

Figura 3.1: Arquivo de IP gerado pelo Choreo no cenário Todos para Todos

IP da máquina 10. Um exemplo deste arquivo é apresentado na Figura 3.1. Este arquivo é enviado para cada uma das máquinas da aplicação.

3.2.1 Choreo

Uma das características do Choreo é utilizar a alocação de mais de uma aplicação ao mesmo tempo para conseguir melhorar a utilização dos links disponíveis. Essa abordagem de escalonamento de mais de uma aplicação ao mesmo tempo faz uso da maior quantidades de máquinas disponíveis para conseguir encontrar uma configuração que melhore as duas aplicações. Mas, em termos de comparação com ClouDiA[25], essa função não possui equivalente e não será utilizada.

Para implementar o Choreo, foi utilizado o pseudo-código disponibilizado no artigo original [13]. Para compreender o pseudo-código é importante definir: P é um conjunto auxiliar de links que armazena os links que serão processados em cada interação do laço de repetição; M é o conjunto de máquinas que enviam algum arquivo, então todos os links terão como origem alguma máquina que pertence a M; N é o conjunto de máquinas que recebem algum arquivo, então todos os links terão como destino alguma máquina que pertence a N; todos os links que pertencem a P terão como origem uma máquina pertencente a M, e como destino uma máquina pertencente a N. Além dessas definições, a *latência* é a média aritmética do tempo de envio de 30 pacotes *ping* entre as máquinas m, n.

Na linha 13 do pseudo-código, existe uma condição que verifica a se a alocação de i em m ou j em n irá exceder a capacidade dos processadores das máquinas m ou n. Pela dificuldade que seria conseguir verificar todas as máquinas a cada iteração, a regra estabelecida foi de alocar

Algorithm 1: Choreo()

```
1  begin
2      transfers = conjunto de tuplas < i, j, b > em ordem decrescente de b. A tupla significa que o
         nó i envia b bytes para j;
3      foreach < i, j, b > em transfers do
4          if i já foi alocada em uma máquina k then
5              P = conjunto de links k → N, ∀ nós N
6          end
7          if j já foi alocada em uma máquina ℓ then
8              P = conjunto de links M → ℓ, ∀ nós M
9          end
10         if i e j não foram alocadas then
11             P = conjunto de links M → N, ∀ nós M e ∀ nós N
12         end
13         foreach link em P do
14             if se a alocação de i em m ou j em n exceder as restrições de CPU de m ou n then
15                 remover m → n de P
16             end
17         end
18         foreach link em P do
19             taxa(m,n) = taxa do link m → n
20         end
21         Alocar i e j em m e n in latência(m,n) seja minimizada
22     end
23 end
```

somente uma tarefa por VM, assim, a conferência é mais simples.

Na linha 19 do pseudo-código, a função *taxa(m, n)* retorna a taxa de conexão vista pelas tarefas caso elas sejam alocadas em m e n. Esse valor seria utilizado na função que encontra quais tarefas serão alocadas, no caso, as máquinas escolhidas seriam aquelas que tivessem o maior valor de taxa de conexão. Esta função não é utilizada neste trabalho, já que a verificação desta informação seria custosa e necessitaria de mais dados.

Para compensar que esta função foi retirada, uma função equivalente foi utilizada, que é a alocação das tarefas no caminho com menor tempo de comunicação. A métrica utilizada difere da original porque o objetivo original era verificar a largura de banda, que é dividida entre todos os processos utilizando o link, e a latência não consegue captar essa informação.

O que acontece na implementação deste método neste trabalho, é que assim que uma das condições é satisfeita, a posição da máquina no vetor já recebe o valor da tarefa. O vetor P que é utilizado no pseudo-código, não é utilizado aqui.

3.2.2 ClouDiA

O ClouDiA utiliza uma abordagem de inicializar mais VMs para aumentar a possibilidade de encontrar melhores conexões, porém, assim como a implantação de mais de uma aplicação por vez do Choreo, neste trabalho não será utilizada esta abordagem. Assim, o número de máquinas utilizadas é igual ao número de tarefas. Além disso, outra mudança realizada foi a metodologia de medição de tempo, que acontece separadamente, e pelo cálculo da média do tempo de retorno da função *ping*.

Para a compreensão do algoritmo, $D(x)$ define uma função que retorna a tarefa que está alocada na máquina x, e $D^{-1}(v)$ define uma função que retorna a máquina virtual onde está alocada a tarefa v. A implementação segue o pseudo-código:

Para obter um algoritmo determinístico, as partes que utilizam escolhas aleatórias foram alteradas. Assim, na linha 5, quando ele diz para encontrar uma aresta arbitrária da aplicação e alocar no link com menor custo do grafo de VMs, a abordagem utilizada foi escolher a aresta mais custosa do grafo da aplicação, assim, as duas tarefas que transferem o maior arquivo serão alocadas.

Outro momento que ele utiliza uma escolha arbitrária é na linha 19, onde ele escolhe *w*, que

Algorithm 2: Cloudia()

```
1  begin
2      S → conjunto de instâncias
3      G = (V, E) → grafo de comunicação da aplicação
4      Encontrar o link u_0 → v_0 de menor custo ∈ S × S
5      Encontrar uma aresta arbitrária (x, y) ∈ G
6      D(x)=u_0
7      D(y)=v_0
8      for i=1 até ||V|| − 2 do
9          c_min=∞
10         foreach (u, v) ∈ S × S do
11             if D^{-1}(v) não foi definido e D^{-1}(u) possui vizinhos não alocados then
12                 if latencia(u, v) < c_min then
13                     c_min = latencia(u, v)
14                     u_min = u
15                     v_min = u
16                 end
17             end
18         end
19         w = um dos vizinhos não alocados de D^{-1}(u)
20         D(w) = v_min
21     end
22 end
```

é um dos vizinhos de u. Neste momento, segundo o pseudo-código, ele pode escolher qualquer um dos vizinhos não alocados. A alteração realizada foi escolher o vizinho com a aresta mais custosa do grafo da aplicação, ou seja, as duas tarefas que transferem o maior arquivo.

Com tais alterações, o algoritmo deixa de ser não determinístico e passa a ser determinístico. Isso foi realizado devido ao impacto que a aleatoriedade teria na comparação dos dois métodos.

Capítulo 4

Resultados

Nesta seção são discutidos os passos necessários para a execução deste trabalho, desde as configurações das máquinas alocadas até a construção dos arquivos utilizados pelas aplicações e os resultados obtidos por cada metodologia de escalonamento.

A Seção apresenta os seguintes tópicos: descrição do ambiente computacional; testes de latência, matriz gerada e comunicação entre zonas; descrição dos grafos de entrada para cada uma das propostas de escalonamento utilizadas; descrição dos experimentos; e resultados.

4.1 Ambiente Computacional

4.1.1 Google Cloud Engine

O Compute Engine é o serviço de infraestrutura (IaaS — Infrastructure as a Service) do Google Cloud Platform. Seu núcleo consiste no uso de máquinas virtuais (VMs — Virtual Machines) de alto desempenho rodando nos data centers do Google e conectadas à sua rede de fibra ótica. Estes recursos estão distribuídos entre diferentes regiões e zonas. Região é uma localização geográfica específica onde você pode executar os seus recursos. Cada região tem uma ou mais zonas, e todos os componentes de hardware são divididos entre as zonas da mesma região. Por exemplo, a região us-central1 denota uma região na região central dos Estados Unidos com as zonas us-central1-a, us-central1-b, us-central1-c e us-central1-f.

Nas instâncias do Compute Engine, é possível executar tanto as imagens públicas para Linux e Windows Server fornecidas pelo Google quanto as imagens particulares que você cria ou importa de seu sistema existente.

A nuvem oferece vários tipos de máquina, que definem uma coleção específica de recursos

de hardware virtualizados disponíveis para uma instância de máquina virtual, incluindo o tamanho da memória, a contagem de CPUs virtuais e a capacidade máxima do armazenamento não volátil. Também é possível criar máquinas com configurações personalizadas.

Para criar e gerenciar instâncias, pode-se usar várias ferramentas, incluindo o console do Google Cloud Platform, a ferramenta de linha de comando gcloud e a REST API. Para executar a configuração avançada, pode-se conectar às instâncias por meio do Secure Shell (SSH) para instâncias Linux ou Remote Desktop Protocol (RDP) para instâncias do Windows.

4.1.2 Configuração das Máquinas

Todas as máquinas possuem a mesma configuração: um processador; 3.95GB de memória RAM; 40GB de armazenamento não volátil. O sistema operacional utilizado foi o Ubuntu 16.04, e as configurações gerais são as configurações padrões da plataforma: escolha de plataforma de CPU automática; preempção das VMs desativada; reinicialização automática em caso de erro; e migração da instância também para tratar erros.

Durante a configuração, também é necessária a escolha da zona da VM. Cada zona possui um limite de VMs do mesmo projeto que podem ser alocadas, o que reforça a heterogeneidade das latências entre os pares de VM em um mesmo projeto. Disponibilizada uma lista com todas as zonas disponíveis, neste trabalho foram escolhidas 3 zonas dentro dos Estados Unidos: 5 máquinas na zona us-east1-b; 6 máquinas na zona us-west1-a; 5 máquinas na zona us-central1-c.

Além destas configurações, foi necessária a liberação da porta 8000 no *firewall* das máquinas, para a comunicação entre os processos das aplicações testadas. E para compilação das aplicações, foram instalados os pacotes *g++* e *pthread* em todas as VMs.

4.1.3 Configuração dos Experimentos

Os testes são realizados em três etapas: a captura dos tempos; a execução dos métodos de escalonamento; a alocação das tarefas e execução das aplicações. O trabalho foi executado da seguinte maneira:

1. Criação e configuração das máquinas virtuais

2. Instalação dos pacotes necessários e liberação da porta 8000 nas configurações do *firewall*

3. Execução dos testes de latência

4. Construção dos arquivos que ditam as comunicações entre os processos nas aplicações

5. Execução dos métodos de escalonamento

6. Configuração e execução de cada cenário de teste

O arquivo que contém os tempos de comunicação entre as máquinas, e os arquivos que contém os grafos que ditam o comportamento das aplicações foram utilizados para a execução dos métodos de escalonamento.

Os arquivos de IP são listas que descrevem o IP onde estão alocadas as tarefas da aplicação. Estes arquivos são resultado dos métodos de escalonamento, portanto variam de método para método. Além do arquivo de IP, para a aplicação Todos para Todos foram utilizados arquivos de execução, que descrevem para que tarefa e a quantidade de quilobytes que serão enviadas, um exemplo deste arquivo é apresentado na Figura 4.1, o primeiro parâmetro é a a quantidade de KB, e o segundo é a tarefa destino.

```
190210 0
87690 1
205920 3
80480  4
293660 5
44210 6
84180 7
212150 8
292720 9
```

Figura 4.1: Arquivo de execução da tarefa 3 da aplicação todos para todos.

4.2 Medição de Latências

Para comprovar a problemática de heterogeneidade de latências entre um conjunto de máquinas virtuais dentro de um mesmo projeto em nuvem, foi calculada a latência entre 10 VMs instanciadas no Google Cloud Engine. Para o cálculo da latência, como já dito anteriormente, foi executada uma série de 30 envios de pacotes *ping* de 64 bytes entre cada par de VM. Para melhorar a demonstração de diferença de tempos, foram utilizadas três zonas para a alocação das máquinas virtuais. As máquinas de 1 até a máquina 3 estão alocadas na zona us-east1-b, que será tratada como zona 1; as máquinas de 4 até a 6 estão alocadas na zona us-west1-a, tratada neste trabalho como zona 2; e as máquinas 7 até 10 estão alocadas na zona us-central1-c, zona 3.

O resultado deste experimento é apresentado na Tabela 4.1, onde é possível verificar uma matriz com os valores em mili segundos da média das 30 execuções. A tabela está dividida por cores, onde cada uma das divisões representa as comunicações entre duas zonas diferentes, assim, cada cor aparece duas vezes, uma onde os pacotes são enviados zona x para a y, e a outra da zona y para a x.

As zonas em azul são os tempos entre VMs da mesma zona, e os itens em amarelo são as médias das latências de comunicação dentro das próprias máquinas.

	1	2	3	4	5	6	7	8	9	10
1	0,037	0,344	0,296	0,04	0,34	0,284	76,155	67,531	67,711	38,965
2	0,345	0,053	0,327	67,657	76,439	76,971	36,169	39,505	39	40,28
3	0,29	0,329	0,035	74,511	75,083	77,212	35,913	36,535	35,887	36,533
4	76,77	81,927	81,798	0,04	0,371	0,407	37,831	36,995	39,32	34,819
5	77,551	78,884	81,861	0,372	0,044	0,36	37,484	39,268	34,755	39,336
6	77,165	81,735	82,068	0,369	0,379	0,039	37,112	37,289	41,877	36,952
7	58,584	35,641	35,666	37,538	36,784	37,677	0,049	0,338	0,328	0,32
8	35,807	36,32	36,263	40,797	34,729	39,713	0,317	0,035	0,305	0,293
9	36,703	35,649	35,586	34,774	34,747	42,113	0,323	0,315	0,035	0,287
10	35,807	36,939	35,558	34,57	37,308	39,648	0,305	0,28	0,318	0,036

Tabela 4.1: Matriz de tempos construída através dos tempos obtidos pelos testes de latência.

Com esta tabela é possível perceber que os tempos de ida e volta podem ser bastante diferentes entre alguns links, tornando assim incorreta a suposição de que o tempo de RTT entre duas máquinas é igual quando se troca a origem e o destino.

Como resumo desta tabela, a Figura 4.2 a seguir apresenta a média dos tempos de comunicação dos tempos utilizados para os experimentos, já que, com a suposição de que os tempos de ida e volta seriam iguais não era necessário utilizar toda a tabela, e foi utilizada somente a matriz triangular superior.

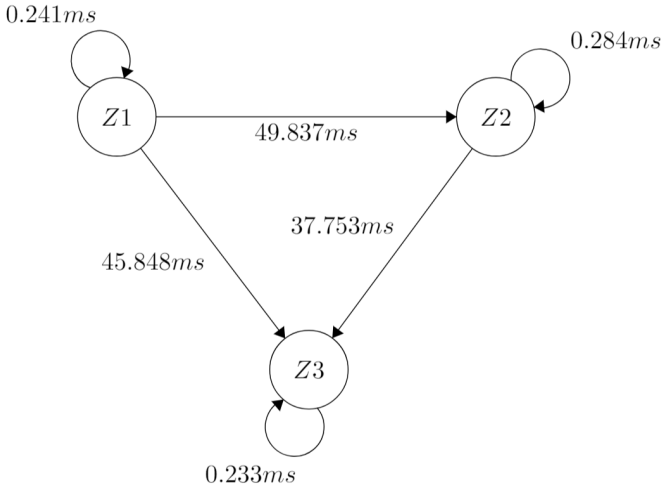

Figura 4.2: Grafo das médias dos tempos de comunicação utilizados nos testes.

4.3 Experimentos

Para a execução dos experimentos deste trabalho, foram utilizadas as topologias Todos para Todos e Cliente-Servidor, as topologias são descritas na Seção 2.1.1. Para cada uma das aplicações, foram executados três cenários: com o escalonamento proposto por Choreo, o escalonamento proposto por ClouDiA, e utilizando a metodologia First Fit, para base de comparação. No escalonamento First Fit, as tarefas foram alocadas em ordem nas máquinas disponíveis, tarefa 1 foi alocada na máquina 1 (zona *east1-b instance-1*), tarefa 2 na máquina 2 (zona *east1-b instance-2*) e assim por diante. Todos os escalonamentos First Fit seguem a Tabela 4.2.

Para cada um destes escalonamentos, uma matriz foi montada aleatoriamente distribuindo valores entre 100B e 100MB. Estes valores seriam utilizados como a quantidade de vezes que a aplicação enviaria um pacote de 1024 bytes para a tarefa destino. Sendo assim, se a entrada é 100, significa que serão enviados 100KB de dados da tarefa *i* para a tarefa *j*. Para evitar o tempo de buscar os arquivos e dados no disco, foram enviados pacotes com lixo, assim, não era necessário abrir, fechar, buscar e ler arquivos.

Tarefa	VM escolhida
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10

Tabela 4.2: Escalonamento utilizando First Fit.

Para medir o tempo de execução das aplicações, foi necessário imprimir o tempo atual (HH:MM:SS) duas vezes em cada um dos processos. O primeiro dentro da função main() dos processos, para captar o horário logo antes de iniciar as funções principais, e o segundo após o laço que realizava o envio dos arquivos presentes no arquivo de execução das tarefas. Assim, o tempo de execução da aplicação foi calculado pela diferença entre o horário de inicio do primeiro processo, e o horário de finalização do último processo de envio.

4.3.1 Todos para Todos

Para a execução desta aplicação, cada uma das 10 tarefas faz o envio de um arquivo para as outras 9 tarefas. Cada uma das tarefas possui um arquivo de execução apresentado na Figura 4.1, que define para onde e quanto de dados enviar. A Figura 4.3 a seguir apresenta uma síntese da comunicação das tarefas.

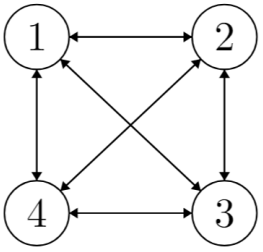

Figura 4.3: Grafo que representa uma síntese das comunicações entre processos em uma aplicação Todos para Todos.

A matriz da aplicação, com todos os valores em KB, é apresentada na Tabela 4.3.

	1	2	3	4	5	6	7	8	9	10
1	0	22392	14732	22925	29951	1463	27160	31299	20235	18766
2	5991	0	16996	24000	25120	3925	24489	4617	20289	7998
3	19021	8769	0	20592	8048	29366	4421	8418	21215	29272
4	11213	20290	28443	0	17367	28668	22261	24704	18027	4917
5	24226	7852	4233	6705	0	16370	9527	21952	14158	15508
6	25758	25569	25386	1157	16255	0	31586	25457	29070	26636
7	1796	13299	7182	12730	30388	1530	0	11513	11065	16815
8	18906	4385	5289	21482	30409	29108	1547	0	27313	1265
9	5638	23029	9358	19908	4753	12426	18540	18064	0	26939
10	7850	2044	18771	32471	23910	16728	24528	2541	3511	0

Tabela 4.3: Matriz da aplicação Todos para Todos. Valores são multiplicados por 10KB.

Utilizando a matriz de tempo, e a matriz de quantidade de dados enviados, foi possível obter o escalonamento da aplicação. Após a execução dos experimentos 5 vezes repetidas, os resultados obtidos a partir de cada um dos escalonamentos é apresentado na Tabela 4.5.

Tarefa	VM Choreo	VM ClouDiA
1	10	10
2	1	6
3	8	9
4	4	4
5	2	7
6	9	3
7	5	2
8	3	1
9	7	5
10	6	8

Tabela 4.4: Tarefa da aplicação, e as VMs escolhidas por cada um dos métodos.

	ClouDiA	Choreo	First Fit
Tempo Médio	78,5	79,33s	136,5s
Comparação com o FF	57,51%	58,12%	100%

Tabela 4.5: Tempo de execução de cada um dos métodos de escalonamento.

A melhoria obtida pelos métodos ClouDiA e Choreo é maior do que 40%, e traz neste caso específico, uma diminuição de 58 segundos no tempo geral da aplicação.

4.3.2 Cliente-Servidor

Para a construção da matriz desta aplicação, foi utilizada a convenção de que o servidor é a tarefa 1. O comportamento da aplicação é apresentado na Figura 4.4 a seguir.

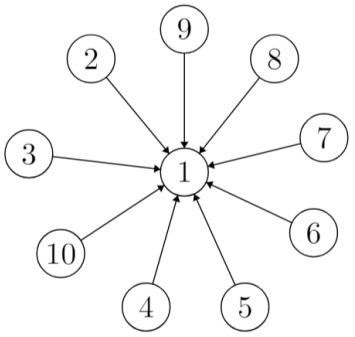

Figura 4.4: Grafo da comunicação entre os processos na aplicação Cliente-Servidor.

A matriz da aplicação é apresentada na Tabela 4.6.

	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	0	0
2	242300	0	0	0	0	0	0	0	0	0
3	40040	0	0	0	0	0	0	0	0	0
4	206780	0	0	0	0	0	0	0	0	0
5	116440	0	0	0	0	0	0	0	0	0
6	86330	0	0	0	0	0	0	0	0	0
7	96850	0	0	0	0	0	0	0	0	0
8	141370	0	0	0	0	0	0	0	0	0
9	105790	0	0	0	0	0	0	0	0	0
10	86790	0	0	0	0	0	0	0	0	0

Tabela 4.6: Matriz da aplicação Cliente-Servidor. Valores em quilobytes.

Utilizando a matriz de tempos e a matriz da aplicação, o escalonamento gerado por ClouDiA e Choreo foram exatamente iguais, e obtiveram uma melhoria de 58,15% no tempo de execução da aplicação.

Tarefa	VM escolhida
1	2
2	6
3	3
4	1
5	4
6	8
7	7
8	9
9	10
10	5

Tabela 4.7: Tarefa da aplicação, e as VMs escolhidas por ambos os métodos.

	ClouDiA	Choreo	First Fit
Tempo Médio	12s	12s	28s
Comparação com o FF	42,85%	42,85%	100%

Tabela 4.8: Tempos de execução da aplicação Cliente-Servidor em comparação com o First Fit.

Nas aplicações Cliente-Servidor e Todos para Todos utilizadas neste trabalho, ambos os métodos de escalonamento apresentaram melhorias significativas, diminuindo mais do que 40% no tempo de execução das aplicações.

Capítulo 5

Conclusão

No contexto de computação em nuvem, a Infraestrutura como Serviço está em constante crescimento. Para lidar com o rápido aumento de usuários que realizam a alocação de suas aplicações distribuídas, os provedores devem realizar melhorias nas infraestruturas que disponibilizam. Apesar de todos os investimentos realizados nas infraestruturas, as arquiteturas de rede em uso atualmente não estão preparadas para lidar com tamanha quantidade de dados [17].

No espectro estudado neste trabalho, as dificuldades da infraestrutura de rede que está sujeita à enormes quantidades de dados trafegados, pode trazer heterogeneidade nos tempos de comunicação entre as máquinas virtuais disponíveis para o cliente. Tais diferenças significativas na latência de comunicação entre as VMs contratadas pelo cliente pode trazer impactos diretos no tempo de execução das aplicações alocadas em nuvem.

Devido à necessidade do provedor de manter sigilo sobre algumas informações, como em que localização (máquinas física ou rack) do *data center* estão localizadas as VMs contratadas, existe uma dificuldade por parte do cliente em obter informações sobre a rede que suas VMs estão utilizando. Sendo assim, quando o cliente realiza a alocação da aplicação, ele não possui os dados necessários para realizar um escalonamento considerando o comportamento da rede entre as máquinas contratadas.

Para lidar com a dificuldade de realizar o escalonamento das aplicações em nuvem, alguns trabalhos disponíveis na literatura foram analisados. Tais estudos propõem métodos para avaliar a heterogeneidade da rede e o comportamento das aplicações que serão alocadas, para assim realizar o escalonamento da aplicação com o objetivo de diminuir o tempo de execução da aplicação.

Foram estudados quatro métodos: CloudTalk [4], LaCurts *et al.* [5], Choreo [13] e ClouDiA [25]. Pela possibilidade de implantação utilizando a infraestrutura atual, foram comparados os métodos que não necessitam de interação com o provedor (Choreo e ClouDiA).

Utilizando estes métodos, e realizando a comparação com o escalonamento First Fit, foram testadas duas aplicações, Todos para Todos e Cliente-Servidor. Ambas as aplicações apresentaram melhorias significativas no tempo de execução das aplicações, a melhoria no escalonamento diminuiu entre 40% e 57,15% no tempo total de execução das aplicações, no contexto de aplicações distribuídas sensíveis à latência, esse tempo tem grande influência no tempo total das aplicações, e na qualidade da usabilidade do usuário final.

Em relação aos dois escalonamentos (Choreo e ClouDiA) não foram encontradas diferenças nas alocações. Por mais que na aplicação Todos para Todos as alocações foram diferentes, o tempo de execução não teve uma diferença considerável, já na aplicação Cliente-Servidor, o escalonamento encontrado por ambos é igual, o que demonstra que a lógica utilizada para escalonar as tarefas é bastante parecida. Pelos experimentos realizados, os métodos são igualmente eficientes nos escalonamentos. Sabendo que as nuvens públicas disponibilizam uma quantidade pequena de informações sobre a arquitetura de rede e das VMS, a capacidade de melhoria utilizando dados que não são coletados continuamente (durante a execução das aplicações) é altamente impactante.

Para trabalhos futuros existem inúmeros pontos a serem estudados, alguns dos principais pontos são: a melhoria na métrica utilizada para medir o tempo de comunicação entre as aplicações; conseguir lidar com as mudanças de carga e comportamento das aplicações dentro de uma parte da nuvem, para evitar que tarefas que foram alocadas fora de *hot spots* sejam colocadas dentro de um; lidar com aplicações reais; encontrar uma maneira de verificar o peso de todas as comunicações realizadas por uma tarefa, e não as comunicações de forma individual.

Referências Bibliográficas

[1] NETWORKING, C. V. *CiscoGlobal Cloud Index: forecast and methodology, 2015-2020.* Consultado na INTERNET: https://goo.gl/zG7Tec, 2017.

[2] STROM, D.; ZWET, J. F. van der. *Truth and lies about latency in the cloud, InterxionTM white paper*. 2015. Consultado na INTERNET: goo.gl/df51gq, 2017.

[3] ZOU, T. et al. Cloudia: a deployment advisor for public clouds. In: *Proceedings of the VLDB Endowment*. Riva Del Guarda, Italy: [s.n.], 2012. p. 121–132.

[4] AGACHE, A.; IONESCU, M.; RAICIU, C. Cloudtalk: Enabling distributed application optimisations in public clouds. In: *Proceedings of the Twelfth European Conference on Computer Systems*. New York, NY, USA: [s.n.], 2017. p. 605–619.

[5] LACURTS, K.; DENG, S.; BALAKRISHNAN, H. *A Plan for Optimizing Network-Intensive Cloud Applications*. Cambrige, USA, Fevereiro 2013.

[6] MELL, P.; GRANCE, T. et al. *The NIST definition of cloud computing*. Gaithersburg, USA, Setembro 2011.

[7] ZOU, T. *Optimizing response time for distributed applications in public clouds*. Tese (Tese de Doutorado) — Cornell University, Janeiro 2015.

[8] ARMBRUST, M. et al. A view of cloud computing. *Communications of the ACM*, ACM, v. 53, n. 4, p. 50–58, 2010.

[9] FERDAUS, M. H. et al. Network-aware virtual machine placement and migration in cloud data centers. *Emerging research in cloud distributed computing systems*, Hershey, PA, EUA, v. 42, p. 42–91, 2015.

[10] WOTTRICH, R.; GENEZ, T.; PEREIRA, W. Uma visão geral sobre sistemas virtualizados. Minicurso de arquitetura de computadores. UNICAMP, São Paulo, Brasil. 2012.

[11] POPEK, G. J.; GOLDBERG, R. P. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, New York, NY, USA, p. 412–421, jul. 1974.

[12] CALCAVECCHIA, N. M. et al. Vm placement strategies for cloud scenarios. In: *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. Honolulu, HI, USA: [s.n.], 2012. p. 852–859.

[13] LACURTS, K. et al. Choreo: Network-aware task placement for cloud applications. In: *Proceedings of the 2013 Conference on Internet Measurement Conference*. New York, NY, USA: [s.n.], 2013. p. 191–204.

[14] BUYYA, R. et al. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, v. 25, n. 6, p. 599–616, Junho 2009.

[15] FERDAUS, M. H. et al. An algorithm for network and data-aware placement of multi-tier applications in cloud data centers. Artigo submetido a avaliação no Journal of Network and Computer Applications (JNCA). Junho 2017.

[16] BALLANI, H. et al. Towards predictable datacenter networks. In: *ACM SIGCOMM Computer Communication Review*. Toronto, Ontario, Canada: [s.n.], 2011. p. 242–253.

[17] GREENBERG, A. et al. Vl2: a scalable and flexible data center network. In: MICROSOFT RESEARCH. *ACM SIGCOMM computer communication review*. Barcelona, Spain, 2009. p. 51–62.

[18] BATTRÉ, D. et al. Evaluation of network topology inference in opaque compute clouds through end-to-end measurements. In: *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. Washington, DC, USA: [s.n.], 2011. p. 17–24.

[19] GOOGLE. *Google Cloud Platform*. Agosto 2017. Consultado na INTERNET: https://cloud.google.com/, 2017.

[20] AMAZON. *Amazon Compute Cloud*. Agosto 2017. Consultado na INTERNET: https://aws.amazon.com/, 2017.

[21] URGAONKAR, B. et al. An analytical model for multi-tier internet services and its applications. In: *ACM SIGMETRICS Performance Evaluation Review*. New York, NY, USA: [s.n.], 2005. p. 291–302.

[22] KUROSE JAMES F.; ROSS, K. W. *Redes de Computadores e a Internet. Uma Abordagem Top Down*. 3. ed. São Paulo, SP: Pearson Education, 2013.

[23] AMAZON. *Amazon Elastic Compute Cloud*. Agosto 2017. Consultado na INTERNET: https://aws.amazon.com/pt/ec2/?ft=n, 2017.

[24] IBM. *IMB ILOG CPLEX Optimizer*. Agosto 2017. Consultado na INTERNET: http://cplex.com, 2017.

[25] ZOU, T. et al. Cloudia: a deployment advisor for public clouds. *The VLDB Journal*, Springer, v. 24, n. 5, p. 633–653, 2015.