Pós-Graduação em Ciência da Computação

PRÍSCILA ALVES LIMA

Restauração de serviços de Data Centers baseada na tomada de decisão de agentes distribuídos

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

Recife
2018

PRÍSCILA ALVES LIMA

Restauração de serviços de Data Centers baseada na tomada de decisão de agentes distribuídos

Este trabalho foi apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: avaliação de desempenho e dependabilidade

Orientador: Dr. Paulo Romero Martins Maciel

Coorientador: Dr. Paulo Roberto Freire Cunha

Recife
2018

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

L732r Lima, Príscila Alves
Restauração de serviços de *Data Centers* baseada na tomada de decisão de agentes distribuídos / Príscila Alves Lima. – 2018.
80 f.: il., fig., tab.

Orientador: Paulo Romero Martins Maciel.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2018.
Inclui referências.

1. Avaliação de desempenho. 2. Tomada de decisão. I. Maciel, Paulo Romero Martins (orientador). II. Título.

004.029 CDD (23. ed.) UFPE- MEI 2019-033

Príscila Alves Lima

Restauração de Serviços de Data Centers Baseada na Tomada de Decisão de Agentes Distribuídos

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Aprovado em: 31/07/2018.

BANCA EXAMINADORA

_____

Prof. Djamel Fawzi Hadj Sadok
Centro de Informática / UFPE

_____

Profa. Rosangela Maria de Melo
Instituto Federal de Ciência de Educação, Ciências e Tecnologia de Pernambuco / Campus Paulista

_____

Prof. Paulo Romero Martins Maciel
Centro de Informática / UFPE

Dedico este trabalho a Deus por seu amor incondicional e pelo cuidado para comigo, à minha família, em especial à minha mãe, por sua dedicação, sua renúncia e seu incentivo que me fizeram chegar até aqui.

AGRADECIMENTOS

A Deus, por sempre estar comigo me dando graça e sabedoria. Foram muitos os desafios para concluir essa etapa da minha vida, mas o Senhor tem me sustentado, tem cuidado de mim e me mostrado que é Ele quem está no controle de tudo: ainda que aos meus olhos a situação seja desfavorável, Ele continua agindo em meu favor**.**

Ao meu orientador Prof. Paulo Maciel, por ter acreditado em mim, pela paciência, pelas suas orientações, pelos conselhos, pela atenção e por sempre responder às minhas dúvidas em um curto espaço de tempo.

À minha querida mãe, Meriam Lima, que mesmo distante fisicamente sempre esteve presente nessa caminhada, me dando forças e me incentivando a continuar, sem seus esforços e renúncia eu jamais estaria aqui hoje.

Aos meus irmãos, Rafael e Sara, pelo carinho e apoio, por compreenderem minha ausência (agora, finalmente, poderei conhecer minha primeira sobrinha, Rafaela).

Ao meu grande amigo Antônio Sá Barreto, por todo apoio, por acreditar em mim, por compartilhar comigo as experiências de seu curso de mestrado. Desde o início, quando a pesquisa estava bem embrionária, sua ajuda foi fundamental para o amadurecimento e desenvolvimento deste trabalho.

Aos meus queridos amigos Carlos, Aretha, Simei, Sérgio, Paty, Ari, Arnon, Elen, Sidney e Fátima (“os manos”), pelo carinho e apoio, apesar da distância. Também deixo aqui meu agradecimento à Patrícia (tetê), que tem sido uma grande amiga desde que cheguei a Recife.

Aos colegas do MoDCS, pela ajuda sempre que precisei e pela companhia nos WMoDCS da vida.

Ao CNPq, pelo auxílio financeiro, que possibilitou a realização deste trabalho.

Enfim, a todos que de forma direta ou indireta contribuíram para esta conquista.

“Em que examinamos métodos para decidir o que fazer hoje, dado que podemos decidir novamente amanhã.”

(NORVIG; RUSSELL, 2014)

RESUMO

Empresas e organizações governamentais estão cada vez mais utilizando soluções em nuvem para prover serviços através da Internet. Essas instituições, atraídas pelos benefícios de confiabilidade, facilidade de manutenção, custo, escalabilidade, agilidade e aumento de produtividade, estão migrando suas rotinas e processos críticos, como aplicações financeiras, comerciais, da indústria e de saúde, para execução em ambiente de nuvem.Com esse crescimento, tem-se também um aumento na implantação de data centers que hospedam toda a infraestrutura de nuvem. Assim sendo, os provedores dessas infraestruturas precisam assegurar o cumprimento do Contrato de Nível de Serviço (SLA), pois a ocorrência de *downtime* pode gerar grandes perdas financeiras para ambas as partes - cliente e provedor. Portanto, a disponibilidade de serviços em nuvem é um fator crítico para os provedores dessa tecnologia, visto que requer uma postura proativa quanto à ocorrência de falhas. Diante desse contexto, este trabalho propõe uma estratégia que usa agentes distribuídos para escolher o link (site) cuja vazão de um data center com defeito para um data center operacional seja mais eficiente em restabelecer o serviço no menor tempo possível. Para atingir este objetivo, desenvolvemos um mecanismo multiagente com três módulos responsáveis por: monitorar os links com o intuito de verificar o *throughput* para transferir uma grande quantidade de dados; rankear a vazão de cada data center com o apoio de agentes e compartilhar informações em uma arquitetura de comunicação distribuída, para que a decisão final possa ser tomada em consenso pelos agentes. Os resultados obtidos comprovam a eficácia da nossa solução, pois foi possível cumprir com os objetivos estabelecidos e fornecer informações para o administrador do data center quanto ao melhor centro de dados para restabelecer serviços (redirecionamento de tráfego) ou realização de migração de VMs.

Palavras-chave: Data center. Tomada de decisão. Agente. Monitoramento. Disponibilidade. Migração de VM.

ABSTRACT

Companies and government organizations are increasingly using solutions deployed in cloud infrastructures to provide services through the Internet. These institutions are migrating their routines and critical processes such as financial, industry and, health applications, to a cloud environment. They are attracted by the benefits which the cloud infrastructure providers offer to their clients such as reliability, ease of maintenance, cost, scalability, agility and, the increase in productivity. This growth forces the cloud service providers to guarantee the Service Level Agreement (SLA) because the downtime can generate significant losses for the companies that deploy their IT infrastructure in the cloud. In this way, the availability of cloud services is a critical factor for the service providers what implies a proactive attitude for failures occurrence. Hence, this work purposes a strategy based on distributed agents to choose a link with better throughput to restore the services of a faulty Data Center in an operational one with the minimal time. To achieve this goal, we developed a multiagent mechanism with three modules responsible for monitor the links to measure the throughput, rank the throughput of each Data Center and, share the information in a distributed architecture in such way which the decision was made in consensus by the agents. The results demonstrate the efficiency of the method, because, at the end of this research we accomplished the stated goals and, provide information to the Data Center administrator about the best DC to restore the services of the faulty one.

Keywords: Data center. Decision making. Agent. Monitoring. Availability. VM migration.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

DC	Data Center
IaaS	Infrastructure as a service.
MTTR	Mean Time to Repair
SLA	Service Level Agreement
VM	Virtual Machine
WAN	Wide Area Network

SUMÁRIO

1 INTRODUÇÃO

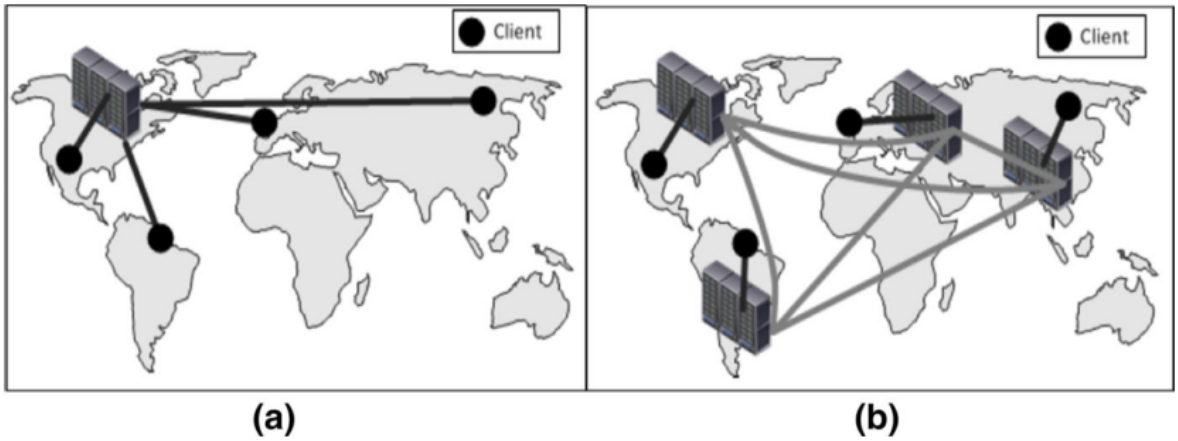
A implantação de infraestrutura para provimento de serviços baseados em nuvem tem crescido nos últimos anos, especialmente por ser constituída por uma robusta plataforma de serviços escaláveis e rentáveis. Essas características são possíveis devido à integração de vários componentes de software, que permitem acesso, sob demanda, a recursos computacionais, através de interfaces e protocolos padrões, baseados principalmente em serviços web (ARMBRUST et al., 2009).

De acordo com Persico et al. (2017), o crescimento de serviços e aplicações entregues através de infraestrutura baseada em nuvem tem atraído um grande número de empresas, cada vez mais dependentes desse ambiente para cargas de trabalho de missão crítica.

O fornecimento de nuvens robustas e confiáveis por inúmeras empresas, como Amazon, Microsoft, Google e IBM, tem motivado os usuários ao uso dessa tecnologia. Os benefícios desse serviço têm contribuído para o crescimento no número de solicitações para o armazenamento de dados na nuvem e, consequentemente, houve um aumento no número de data centers para atender a essa demanda. Um dos serviços de nuvem, o Google, possui 36 *Data Centers* (DC) distribuídos no mundo, sendo 19 na América, 12 na Europa, 3 na Ásia, 1 na Rússia e 1 na América do Sul (CHEN; GAO; CHEN, 2016) .

Alguns provedores de nuvem implantam seus data centers em localizações geográficas específicas (topologia centralizada), enquanto outros distribuem seus data centers ao redor da terra (topologia distribuída), conforme pode-se verificar no exemplo da Figura 1 (ZIAFAT; BABAMIR, 2017).

Figura 1 – Exemplo da topologia de data centers centralizado (a); exemplo da topologia de data centers distribuído (b).



Fonte: Ziafat (2017)

Atrelado a esse crescimento**,** tem-se um fator crítico para os provedores de serviços em nuvem**:** a disponibilidade, pois é necessário cumprir com o Contrato de Nível de Serviço (SLA - Service Level Agreement) mesmo em casos de falhas. Como exemplo de problemas com disponibilidade de serviço em nuvem, pode-se citar o evento ocorrido em fevereiro de 2017, quando um data center da Amazon S3 (Simple Storage Service) ficou inoperante por algumas horas e deixou várias empresas e usuários com acesso indisponível nos Estados Unidos e em todo o mundo (LEE, 2017). Segundo a Amazon a inoperância aconteceu devido a uma falha humana, em que um funcionário autorizado ao executar um procedimento de rotina para retirar um pequeno número de servidores do S3, digitou um comando incorreto que resultou na retirada de um número maior que o esperado de servidores (AMAZON, 2017).

As falhas são inevitáveis no mundo real, portanto, é prudente adotar projetos tolerantes a falhas, contemplando estratégias e procedimentos para mitigar tais eventos, com tempo médio de reparo (MTTR) mínimo e**,** consequentemente**,** interrupção mínima do serviço. Desse modo, o planejamento de continuidade de negócios envolve um plano de contingência para atenuar riscos catastróficos através de planos de continuidade que garantam que as operações possam ser recuperadas prontamente após um desastre natural ou provocado pelo homem (BAUER; ADAMS; EUSTACE, 2011) (ISO/IEC-27002, 2005)(MACIEL, 2016).

Diante disso, estabeleceu-se então a seguinte pergunta de pesquisa:

- *Como auxiliar no processo de tomada de decisão para escolha dos data centers com melhor vazão para restaurar serviços de um data center falho com base na análise do link?*

Para responder a essa pergunta, foi feito um estudo na literatura para identificar trabalhos existentes que visassem minimizar o tempo de reparo e que utilizassem estratégias de migração de VMs na WAN. Esse levantamento teve por finalidade buscar ideias que corroborassem com a construção da proposta deste trabalho, conforme será discutido no próximo capítulo.

1.1 Motivação e Justificativa

De acordo com a Gartner (2017) o mercado de nuvem pública de infraestrutura como serviço (IaaS) cresceu 31% em 2016, totalizando US$ 22,1 bilhões. A Amazon foi o fornecedor número um no mercado IaaS neste mesmo ano, seguido pela Microsoft e Alibaba. Para Sid Nag, diretor de pesquisa da Gartner, a demanda por IaaS continua crescendo agressivamente e isso deve se intensificar nos próximos cinco anos.

Um estudo realizado pelo IDC (2017), prevê que os gastos com nuvem pública atingirão US$ 203,4 bilhões em todo o mundo até 2020. Esse crescimento se dá pela migração de empresas e organizações para ambientes de nuvem, possivelmente atraídos pelos benefícios que tal tecnologia oferece.

Nesse cenário, em que cada vez mais empresas estão migrando para ambientes de nuvem, é interessante definir estratégias para garantir a continuidade dos serviços**,** pois eventos como terremotos, inundações, incêndios, falhas humanas ou de software/hardware podem ocorrer. Dessa forma, é prudente prover mecanismos que tratem as interrupções dos serviços, uma vez que o custo do downtime pode impactar negativamente nos negócios (ANDRADE, 2014).

Visto, então, o grande crescimento da utilização de infraestrutura em nuvem, e considerando o quão críticos são esses serviços, de modo que requerem que os provedores adotem estratégias proativas para minimizarem o downtime na ocorrência de falhas, faz-se necessário adotar alternativas que mitiguem esses riscos. Este trabalho visa buscar algumas dessas alternativas usando uma estratégia de medição ativa da rede, visando à construção de um mecanismo baseado em sistemas multiagentes, capaz de auxiliar os administradores de data centers no processo de decisão para restauração dos serviços no menor tempo possível.

1.2 Objetivos

Esta pesquisa se concentra na análise de condições de rede para medir a vazão de links que interligam um conjunto de data centers geograficamente distribuídos. Através destas medições, pretendemos escolher o data center cujo link seja mais eficiente e para o qual as requisições de um data center com falha possam ser redirecionadas.

A estratégia adotada também pode ajudar na escolha do melhor DC a fim de migrar máquinas virtuais (VMs) para fins de backup distribuído.

Analisar previamente condições de rede para então decidir para qual site realizar a migração de VM ou o redirecionamento de fluxo pode ser de grande valia para uma escolha assertiva. Segundo Arif, Kiani e Qadir (2016) , os links da WAN geralmente são imprevisíveis e limitados em largura de banda devido à interferência de alta latência, jitter e perda de pacotes.

Diante dessas informações, foi estabelecido o seguinte objetivo geral para esta pesquisa:

- Construir um mecanismo multiagente para dar suporte no processo de tomada de decisão para escolha do data center no qual serão restaurados os serviços de um DC falho ou para fins de backup de VMs.

Por mecanismo, entende-se como uma aplicação computacional que envolve um conjunto de ferramentas e procedimentos capazes de mensurar condições de rede e apresentar ao administrador de sistemas as opções de possíveis data center com melhores condições de vazão para migração de serviços.

As instalações de rede e os equipamentos de acesso localizados fora dos data centers protegidos, segundo Bauer, Adams e Eustace (2011), podem ser vulneráveis a mais riscos e são mais propensos a falhar; portanto, exigirão atenção especial. Dessa forma, para minimizar as penalidades de violação de SLA, ações pró-ativas, como o monitoramento da rede, são necessárias para escolher um data center com throughput mais eficiente para restaurar os serviços de um DC com falha.

Assim, foram estabelecidos os seguintes objetivos específicos para esta pesquisa:

- Utilizar métodos experimentais para estudar o impacto de parâmetros de rede sobre a vazão.
- Definir um procedimento de análise das condições de rede**.**
- Construir agentes para auxiliar no processo de tomada de decisão**.**
- Implantar uma arquitetura de comunicação entre os data centers geograficamente distribuídos.

1.3 Contribuições

As principais contribuições deste trabalho são:

- Identificação de ferramental para monitoramento de rede de longa distância (WANs)**.**
- Identificação de estratégia para comunicação distribuída em um sistema multiagente**.**
- Desenvolvimento de um algoritmo capaz de *rankear* os servidores de acordo com a melhor vazão do link, dado um *pool* de data centers geograficamente distribuídos**.**
- Desenvolvimento de um mecanismo que apoia a tomada de decisão para escolha do data center que melhor atenda, no que diz respeito ao throughput do link, a um determinado data center em condição de falha.

1.4 Estrutura da Dissertação

Esta dissertação está organizada em 7 capítulos. Sendo o presente **Capítulo 1** a Introdução, e os demais conforme apresentado a seguir**.**

Capítulo 2: Os trabalhos relacionados

Capítulo 3: Fundamentação teórica, apresentando os conceitos utilizados neste estudo

Capítulo 4: Solução Proposta

Capítulo 5: O estudo de caso

Capítulo 6: Considerações finais e trabalhos futuros.

2 TRABALHOS RELACIONADOS

Neste capítulo, é apresentado um conjunto de trabalhos relacionados que foram selecionados através de um levantamento bibliográfico com o intuito de coletar e analisar estudos que abordassem técnicas existentes para migração de VMs na WAN, além de apoiar a tomada de decisão para escolha de data center geo distribuído considerando sua vazão.

2.1 Procedimento para seleção dos trabalhos

A seleção dos trabalhos acadêmicos deu-se buscando responder às perguntas abaixo mencionadas.

- *Quais as estratégias adotadas para migração de máquinas virtuais entre data centers geograficamente distribuídos com menor custo de tempo?*
- *Como apoiar o administrador de sistema na tomada de decisão para migração de máquinas virtuais considerando as condições da rede?*

Após essa triagem inicial, o levantamento bibliográfico culmina em cinco questões de investigação mais específicas que possibilitam responder a tais problematizações:

- (Q1) Quais os principais desafios para migração de VMs em um ambiente de data center distribuído?
- (Q2) Quais as melhores práticas a serem adotadas no processo de migração de VMs em ambiente de data center distribuído?
- (Q3) Que ferramentas existem para fazer análise de rede a fim de auxiliar no processo de tomada de decisão para migração de VMs inter data centers?
- (Q4) Que estratégias existem para realizar a escolha de rotas para migração de VMs entre data centers de forma aware (consciente)?
- (Q5) Quais as ferramentas e modelos para gerenciamento de migração de VM?

Inicialmente, foram coletados 50 artigos no Google Scholar, de modo a possibilitar a identificação das palavras-chave mais recorrentes e auxiliar a construção das Strings de Busca. Para cada questão, foi definida uma string de busca, conforme apresentado na Tabela 1 **.**

Tabela 1 – Stringsde Busca dos artigos

Questão	Strings de Busca
Q1	(“Challenge” OR “Difficult” OR “Critical Factor” OR “Problem”) AND (“Cloud Data Center” OR “Geographically distributed data centers” OR “Geo-distributed data centers” OR “inter-DCs communication”) AND (“Virtual machine” OR “VM”) AND (“VM migration” OR “Migration” OR “Live Cloud Migration” OR “Network-aware virtual machine Migration”)
Q2	(“Migration Strategy” OR “Migration mechanisms” OR “Migration method”) AND (“Cloud Data Center” OR “Geographically distributed data centers” OR “Geo-distributed data centers” OR “Inter-DCs communication”) AND (“Virtual machine” OR “VM”) AND (“VM Migration” OR “Migration” OR “Live Cloud Migration” OR “Network-aware virtual machine Migration”)
Q3	(“Tool” OR “Software” OR “Program” OR “System”) AND (“Cloud Data Center” OR “Geographically distributed data centers” OR “Geo-distributed data centers” OR “Inter-DCs communication”) AND (“Virtual machine” OR “VM”) AND (“Model” OR “Framework” OR “Method” OR “Technique” OR “Methodology”)
Q4	(“Cloud Data Center” OR “Geographically distributed data centers” OR “Geo-distributed data centers” OR “Inter-DCs communication”) AND (“Virtual machine” OR “VM”) AND (“VM Migration” OR “Migration” OR “Live Cloud Migration” OR “Network-aware virtual machine Migration”) AND (“Challenge” OR “Difficult” OR “Critical Factor” OR “Problem”) AND (“Migration Strategy” OR “Migration mechanisms” OR “Migration method”) AND (“Tool” OR “Software” OR “Program” OR “System”) AND (“Model” OR “Framework” OR “Method” OR “Technique” AND “Methodology”)
Q5	(“Cloud Data Center” OR “Geographically distributed data centers” OR (“Geo-distributed data centers” OR “Inter-DCs communication”) AND (“Virtual machine” OR “VM”) AND (“VM Migration” OR “Migration” OR “Live Cloud Migration” OR “Network-aware virtual machine migration”) AND (“Tool” OR “Software” OR “Program” OR “System”) AND (“Model” OR “Framework” OR “Method” OR “Technique” OR “Methodology”) AND (“Management”)

As etapas do processo do levantamento bibliográfico são apresentadas na Figura 2.

Figura 2 – Etapas do processo de seleção dos trabalhos relacionados

A **Estratégia de Busca** dos trabalhos relacionados seguiu alguns critérios para a seleção dos estudos mais relevantes. As buscas ocorreram no período de março a junho de 2017 e as bases de dados eletrônicas consultadas foram: *Springer Link, Engineering Village, ScienceDirect* e *Scopu*s. Na etapa de **Coleta**, para cada fonte de busca foram aplicadas todas as strings descritas na Tabela 1, resultando num total de 200 trabalhos coletados. Em seguida, na fase de **Triagem**, buscou-se excluir os estudos que estavam fora do escopo desta pesquisa, uma vez que o principal objetivo desta pesquisa é buscar estratégias de migração através do monitoramento de rede.

Após esse filtro, restaram 50 trabalhos que passaram por uma análise crítica. Por fim, na etapa de **Seleção dos Trabalhos**, dentre todos os estudos analisados, apenas 10 artigos foram selecionados por estarem diretamente relacionados a esse estudo ou por contribuírem, em algum grau de relevância, com o desenvolvimento desta pesquisa.

A análise dos trabalhos coletados revelou que a maioria deles se concentra na migração de VMs dentro dos data centers e poucos consideram o desempenho da rede para migração em redes de alta latência. Essa informação pode ser corroborada com o trabalho de Biswas et al. (2016) que também chegou a essa conclusão.

Os pesquisadores propuseram várias técnicas de migração para VMs, como Live Migration, que são efetivas em uma LAN. No entanto, quando se trata de WAN, as condições da rede mudam e essas técnicas não são mais aplicáveis, o que pode causar um tempo de inatividade significativo, fazendo o processo de migração falhar (KAPIL; PILLI; JOSHI, 2013).

Os trabalhos discutidos nesta seção também demonstram o interesse da comunidade acadêmica no uso de aprendizagem de máquina para automatizar o processo de monitoramento e tomada de decisão na nuvem, migração de dados na WAN e a importância de monitorar a infraestrutura de nuvem para garantir a disponibilidade e a conformidade com o SLA.

Zhang, Liu e Chen (2016) propuseram um *framework* de balanceamento de carga distribuído baseado em múltiplos agentes para plataforma em nuvem. A solução

utiliza séries temporais para fazer predições de carga de trabalho, estratégias de *buffering* e limiar para reduzir a quantidade de migrações de VMs.

O *framework* possui cinco módulos: *1-Monitoramento* (coleta e armazena os dados das máquinas físicas, virtuais e o histórico da carga de trabalho); *2-Predição* (prevê a carga de trabalho futura de acordo com os dados históricos da carga de trabalho da máquina); *3-Tomada de decisão* (decide se deseja migrar ou não a VM e escolhe qual máquina deve ser migrada); *4-Licitação* (quando o gerenciador de VMs precisa executar uma migração, ele envia uma mensagem para cada agente inteligente localizado nas máquinas físicas, e os agentes precisam dar um lance. Essa informação auxiliará o gerenciador de VM, que escolherá o maior lance como *host* de destino da migração); *5-Envio e recebimento de mensagem* (é a interface unificada entre o agente e o exterior).

As estratégias utilizadas nesse estudo minimizam a quantidade de carga de trabalho resultante da transmissão de dados, reduzindo a carga do gerenciador de VMs e aumentando a confiabilidade do data center. No entanto, esse estudo limita-se à migração dentro do data center.

Persico et al. (2017) realizaram uma análise aprofundada para verificar o desempenho de redes de ampla abrangência, que interconectam nós de nuvens geograficamente distribuídas, considerando os provedores da Amazon Web Services e Azure. O impacto da rede foi verificado por vários fatores de configurações sob controle dos consumidores desses serviços, a região e o tamanho das VMs usadas. Eles concluíram que a infraestrutura dos data centers da Azure supera a da Amazon em termos de *throughput*. Resultados empíricos mostraram que os provedores dependem de suas próprias infraestruturas dedicadas para conectar locais geograficamente distribuídas. No entanto, sob certas circunstâncias, eles também dependem da infraestrutura de rede de terceiros; dessa forma, os provedores são forçados a fornecer aos seus consumidores um serviço com baixo desempenho e altos custos.

Esse estudo evidencia a importância de considerar condições de rede ao projetar estratégias para migração de dados na WAN, uma vez que diversos fatores podem impactar na sua vazão devido à sua natureza dinâmica.

O trabalho de Teyeb et al. (2016) aborda o problema do agendamento para migração de VM em centros de dados geograficamente distribuídos. A estratégia proposta pelos autores visa otimizar as decisões de posicionamento da VM, minimizar o número de migrações e reduzir o tráfego entre os data centers, evitando o congestionamento dos links. Eles propuseram soluções exatas e heurísticas considerando a vida útil das VMs e os experimentos mostraram que o tempo de vida das VMs afeta a tomada de decisão de migração. Os autores demonstraram a eficácia da solução em termos da estabilidade do sistema.

A pesquisa realizada por eles é relevante, pois preocupa-se com a redução de migração de VMs para evitar congestionamento de link. No entanto, eles não consideram o desenvolvimento de estratégias para redução do impacto na ocorrência de falhas.

Duggan, Duggan e Barrett (2016) afirmam que a capacidade de executar a migração de VM usando a estratégia *live migration* pode ter um grande impacto na forma como um sistema em nuvem é executado, principalmente porque pode consumir quantidades significativas de recursos de rede, como largura de banda. Os autores propuseram uma estratégia baseada em uma técnica de inteligência artificial conhecida como *Reinforcement Learning* para executar a migração de VM de forma autônoma com a consciência do tráfego de rede.

A estratégia utiliza agentes autônomos capazes de aprender e decidir um horário apropriado para agendar a migração de VMs, analisando a disponibilidade da largura de banda do data center. Todavia, este estudo limita-se à migração dentro do data center, não contemplando análise de rede de ampla abrangência.

Outro estudo selecionado foi o de Diallo et al. (2016)**,** que investiga a disponibilidade de serviços em nuvem. Nele, os autores propuseram um *framework* inteligente que monitora e executa serviços de *Live Migration* na nuvem em caso de ameaças de interrupção. A solução inclui algoritmos de seleção que priorizam os serviços críticos, identificam quais serviços devem ser migrados, quando precisam ser migrados e para onde esses serviços devem ser migrados.

Para isso, foi desenvolvido o *framework* **AutoMigrate**, que combina um conjunto de componentes (Monitor de VM, Detector de Anomalia, Seletor do Serviço em Nuvem, Seletor de destino e Gatilho de migração), criando um sistema autônomo que tem ciência a respeito: do estado das VMs, da importância dos serviços hospedados em máquinas virtuais e das dependências entre serviços. Os pesquisadores desenvolveram um interessante trabalho, criando uma ferramenta que se antecipa à ocorrência de falhas, porém limita-se à execução na LAN do data center.

No estudo de Ziafat e Babamir (2017), os pesquisadores propõem um método para selecionar o melhor centro de dados em um conjunto de DCs geo distribuídos em relação ao SLA e aos requisitos do usuário, com base em quatro atributos de qualidade: disponibilidade, confiabilidade, custo e tempo de resposta. Na abordagem proposta, os data centers são agrupados de acordo com esses atributos, os quais são classificados como critérios negativos ou positivos. O algoritmo NSGA-II multi-objetivo é usado para a seleção do data center ideal de acordo com tais atributos de qualidade e os autores desenvolveram o método NSGAII *Cluster*, uma combinação dos algoritmos k-means e NSGA-II. O método foi comparado com outros três algoritmos (Greedy, Random e MOPSO) e conseguiu selecionar o melhor data center de um conjunto de centros de

dados geo distribuídos de acordo com as regras do SLA e a solicitação do usuário.

Esse estudo é bastante relevante pois extrapola as redes locais indo para WANs, além de preocupar-se com atributos de qualidade e garantia do cumprimento do SLA. Mas, está fora de seu escopo o monitoramento da rede e a seleção do data center com melhor vazão na ocorrência de falhas.

No estudo de Xu, Chen e Calero (2016), os autores propuseram uma arquitetura de monitoramento para infraestrutura em nuvem. Eles argumentaram que os modelos de monitoramento centralizados levam facilmente a um único ponto de falha. Este problema foi resolvido desenvolvendo um modelo de monitoramento colaborativo distribuído (DCMM) e um algoritmo de controle de limiar adaptativo (ATCA).

O DCMM adota a topologia de anel distribuído na arquitetura, no qual um componente conhecido como Nó de Monitoramento lógico permanece no centro do anel com vários Nós de dados/Nós Monitorados (DNs) na borda. No entanto, as tarefas de monitoramento são igualmente compartilhadas entre os DNs, enquanto a ATCA visa reduzir os dados de monitoramento redundantes produzidos pela arquitetura de monitoramento. Comparado com os modelos de monitoramento centralizado, a solução proposta demonstrou que pode resolver os problemas associados a um único ponto de falha e desempenho do modelo de monitoramento centralizado. Destacamos que esse trabalho realiza monitoramento distribuído, porém não contempla o monitoramento de um conjunto de data centers geo distribuídos.

Arif, Kiani e Qadir (2016) propuseram uma estratégia de migração para links WAN. Nesse artigo, os autores apresentam o MLDO, uma abordagem que emprega métodos de predição e aprendizado automático para fazer *live migration* e reduzir o *downtime*. A abordagem contém dois módulos que funcionam em cooperação: Mecanismo de monitoramento e o Mecanismo de processamento. O primeiro é responsável pelo monitoramento de parâmetros físicos e virtuais, tais como CPU, memória, utilização da rede e registro de dados no banco de dados para análise heurística.

O mecanismo de processamento é usado para comparar os dados com limiares predefinidos e aplicar métodos de aprendizagem de máquina para tomar decisões – a idéia é iniciar a migração de VMs com base em seus padrões de uso, bem como as condições de carga no servidor físico. De acordo com os pesquisadores, foi possível, com o MLDO, reduzir o tempo de inatividade observado durante o processo de migração em até 15%, quando comparado com outras abordagens.

Esse trabalho é relevante para o contexto desta pesquisa, pois ele cria uma estratégia que se preocupa com a minimização do tempo de inatividade nas migrações em links WANs. Ele não foca no monitoramento e identificação da melhor vazão de data centers, mas propõe um método de migração que pode ser agregado ao mecanismo

proposto na presente pesquisa.

Lu et al. (2016) desenvolveram um sistema de monitoramento de nuvem chamado JTangCMS. O sistema é capaz de coletar, entregar e processar os dados monitorados através de agentes. Além disso, também é implementada uma estrutura de disseminação de dados eficiente e robusta para transferir grande quantidade de informações de tempo de execução de forma confiável. O *framework* dar suporte à tomada de decisões para o gerenciamento da nuvem com base no processamento complexo de eventos (CEP). A proposta desse trabalho é interessante e apresenta um *framework* que se mostra robusto para o monitoramento de nuvem. No entanto, ele se concentra apenas dentro do data center.

Jammal et al. (2016) abordam a disponibilidade de serviços em nuvem e desenvolvem uma abordagem *HA-aware* (consciência da alta disponibilidade) para mitigar os riscos de *downtime*. Além disso, os autores propõem uma técnica de *live migration* visando manter a entrega do serviço após uma falha súbita de VM/sobrecarga de infraestrutura ou manutenção. Para isso, desenvolveram um modelo de otimização utilizando programação linear inteira mista (MILP) que minimiza o tempo de inatividade da migração com base na página de memória da VM e na colocação *HA-aware* ideal da VM.

Até o presente momento não é do nosso conhecimento a existência de trabalhos que utilizem a abordagem empregada nesta pesquisa. Uns dos trabalhos que mais se aproximam deste estudo são os de Zhang, Liu e Chen (2016) e Lu et al. (2016), porém eles se concentram em redes locais. A Tabela 2 resume as principais contribuições desta pesquisa e em que ela se diferencia dos demais trabalhos relacionados apresentados. Os critérios listados na tabela estão relacionados a seguir.

Critério 1: Monitoramento distribuído de rede

Critério 2: Análise de tráfego de rede na WAN

Critério 3: Solução com agente/multiagente

Critério 4: Tomada de decisão distribuída

Critério 5: Garantia da disponibilidade

Tabela 2 – Tabela comparativa dos trabalhos coletados

Trabalho	Contexto	Critério 1	Critério 2	Critério 3	Critério 4	Critério 5
Zhang *et al.* (2016)	Balanceamento de carga baseado em agentes			√	√	
Teyeb *et. al.* (2016)	Problema de agendamento para migração de VM		√			
Duggan *et al.* (2016)	Migração de máquina virtual com reconhecimento de rede	√		√		
Diallo *et al.* (2016)	*Live migration* para garantia da disponibilidade				√	√
Xu *et al.* (2016)	Monitoramento distribuído para infraestruturas de cloud	√			√	
Lu *et al.* (2016)	Monitoramento de cloud através de agentes	√		√		
Jammal *et al.* (2016)	*Live migration* para garantia da disponibilidade					√
Arif *et al.*(2016)	*Live Migration* em links WANs	√	√			
Persico *et al.* (2017)	Análise e monitoramento de rede em infraestrutura na WAN	√	√			
Ziafat *et al.* (2017)	Seleção de DC com base em atributos de qualidade				√	√
Esta pesquisa		√	√	√	√	√

Os trabalhos escolhidos durante o levantamento bibliográfico foram de suma importância para compreensão e estruturação da pesquisa, pois forneceram subsídios dos pontos que poderiam ser explorados. No próximo capítulo, discutiremos os principais conceitos envolvidos para o desenvolvimento desta pesquisa.

3 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os conceitos fundamentais para o entendimento deste trabalho. Inicialmente, serão apresentados os conceitos de alguns parâmetros de rede que são utilizados para mensurar a performance da mesma. Em seguida, discutiremos sobre o planejamento de experimento, que consiste em uma técnica para verificar a influência de determinados fatores no sistema. Além disso, serão apresentados os conceitos sobre agentes, consenso distribuído para tomada de decisão e, por fim, mecanismo de troca de mensagem.

3.1 Métricas de desempenho de rede

De maneira geral, seria interessante que os serviços da Internet transferissem tantos dados quanto desejássemos entre dois sistemas finais, instantaneamente, sem nenhuma interferência. Porém, isso é algo inalcançável, pois as leis físicas da realidade introduzem atrasos, perdas, assim como a restrição na vazão (KUROSE; ROSS, 2010).

Algumas medidas de desempenho de uma rede são comumente definidas através de métricas tais como: latência (*Delay*), variância estatística do tempo de chegada dos pacotes (*Jitter*), número de bits que um link é capaz de transmitir (Largura de Banda) e a capacidade de fluxo (*Throughput*).

3.1.1 Delay

A latência ou retardo define quanto tempo leva para uma mensagem inteira chegar de forma completa no seu destino, desde o momento em que o primeiro bit é enviado da origem (FOROUZAN, 2007). As ferramentas comuns usadas para medir o tempo de requisição usam o *Round Trip Time* (RTT), através do RTT podemos obter o Delay usando a relação mostrada na Equação 3.1.

$$D = RTT/2 \tag{3.1}$$

3.1.2 Jitter

Jitter refere-se à variação no tempo de chegada entre os pacotes. Para estimar o Jitter, utiliza-se a relação definida por muitos autores em redes de computadores, conforme explicado por Toncar (2018) e apresentada na Equação 3.2, cujos termos são explicados na Tabela 3 .

$$J_i = J_{i-1} + (|D_i - D_{i-1}| - J_{i-1})/16 \tag{3.2}$$

Tabela 3 – Significado dos termos da equação

Termo	Significado
J_i	Jitter no instante *i*
$Ji - 1$	Jitter no instante anterior *i - 1*
Di	Delay no instante *i*
$Di - 1$	Delay no instante *i - 1*

3.1.3 Largura de Banda

Uma outra característica que mede o desempenho das redes é a largura de banda. Ela se refere ao número de bits por segundo que um canal, um enlace ou até mesmo uma rede é capaz de transmitir. Por exemplo, pode-se dizer que a largura de banda de uma rede Ethernet (ou os enlaces nessa rede) é de no máximo 100 Mbps. Isso significa que essa rede pode enviar 100Mbps (FOROUZAN, 2007).

3.1.4 Throughput (Vazão)

Esta métrica é uma medida da rapidez pela qual podemos realmente enviar dados pela rede. Embora, à primeira vista, a largura de banda em bits por segundo e *throughput* pareçam a mesma coisa, eles são diferentes. Um enlace pode ter uma largura de banda de B bps, mas podemos enviar apenas T bps por esse enlace, em que T é sempre menor que B. Em outras palavras, a largura de banda é uma medida possível de um enlace; o *throughput* é uma medida real da rapidez pela qual podemos enviar dados. Poderíamos ter, por exemplo, um enlace com largura de banda de 1 Mbps, mas os dispositivos conectados na extremidade do enlace seriam capazes de lidar com apenas 200 kbps. Isso significa que não podemos enviar mais de 200 kbps por esse enlace (FOROUZAN, 2007). De acordo com a RFC 1242 trata-se da taxa máxima em que nenhum dos *frames* oferecidos é descartado pelo dispositivo (GROUP, 1991).

Embora o *delay* e a largura de banda sejam duas métricas de desempenho, a relação mais importante usada na área de redes para medir o impacto delas no tempo de transmissão de dados é o produto entre elas. Assim, o produto entre largura de banda e atraso define o número máximo de bits para preencher o link.

3.2 Planejamento de Experimento

O Planejamento de Experimento é um método utilizado para avaliar o impacto de cada parâmetro de um determinado sistema. De acordo com Jain (1990), seu objetivo é obter o máximo de informações com o mínimo de experimentos, reduzindo o trabalho com a coleta dos dados, além de facilitar a separação dos efeitos de vários fatores que podem influenciar o desempenho.

Para projetar um experimento é necessário definir fatores, níveis e uma variável de resposta. Através dessa técnica, é possível averiguar a influência dos fatores sobre a variável de resposta, e, para analisar os efeitos de cada fator na variável de resposta, é necessário obter valores que representam os níveis desses fatores (JAIN, 1990).

Com este método, é possível avaliar a importância de cada um dos parâmetros (fatores) do sistema e, além disso, ele pode ser usado para determinar simultaneamente os efeitos individuais e interativos de fatores que podem afetar as medidas de saída (MATHEWS, 2005).

Existem vários tipos de planejamento de experimento e os mais usados são: planejamento simples, planejamento fatorial completo e planejamento fatorial fracionário (JAIN, 1990).

- No planejamento simples, inicia-se com uma configuração típica e varia-se um fator por vez, para ver como esse fator afeta o desempenho. No entanto, esse método não faz o melhor uso do esforço gasto. Não é estatisticamente eficiente. Além disso, se os fatores tiverem interação, esse método pode levar a conclusões erradas (JAIN, 1990).

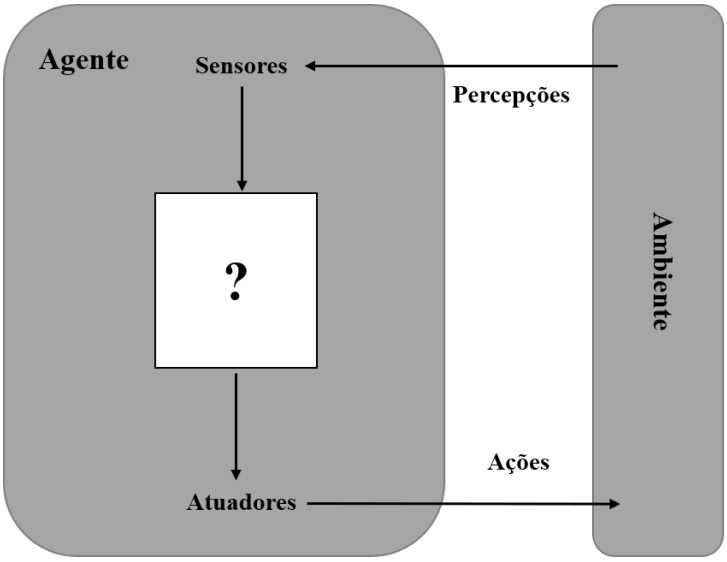
- Um planejamento fatorial completo exige muitos experimentos, pois todas as combinações possíveis de configuração e carga de trabalho são examinadas. Considerando um sistema, no qual o seu desempenho é afetado por k fatores, e cada fator tem N níveis possíveis de valores, o número de experimentos a serem executados é calculado por N^k (MATHEWS, 2005) (JURISTO; M. MORENO, 2013).

- O planejamento fracionário economiza tempo e dinheiro, mas fornece menos informações que o método completo. Por exemplo, pode-se obter algumas, mas não todas as interações entre os fatores. Fatoriais fracionários são comumente usados para reduzir o número de execuções necessárias para construir um experimento. Eles são recomendados para os casos em que o experimento possui 2^K modelos com cinco ou mais variáveis (MATHEWS, 2005) (JURISTO; M. MORENO, 2013).

3.3 Agentes

Um agente pode ser definido como um sistema informático que está situado em algum ambiente e é capaz de executar ação autônoma nesse ambiente para atingir os objetivos do projeto. Assim, um agente é tudo que pode ser capaz de perceber o meio ambiente através de sensores e atuar sobre esse ambiente através de atuadores, conforme pode-se verificar na Figura 3 (WOOLDRIDGE, 2009) (RUSSELL; NORVIG; DAVIS, 2016).

Figura 3 – Interação do agente com o ambiente



Fonte: Russell et al (2016)

O termo percepção é usado para fazer referência às entradas perceptivas do agente em um dado instante. A sequência de percepções do agente é a história completa de tudo o que o agente já percebeu. Em geral, a escolha de ação de um agente em um dado instante pode depender da sequência inteira de percepções recebidas até o momento, mas não de percepções não recebidas. Se pudermos especificar a escolha de ação do agente para toda sequência de percepções possível, teremos dito quase tudo o que existe a dizer sobre o agente. Em termos matemáticos, afirmamos que o comportamento do agente é descrito pela função do agente que mapeia qualquer sequência de percepções específica para uma ação (RUSSELL; NORVIG; DAVIS, 2016).

Os agentes geralmente são orientados por utilidade que são funções que medem o benefício obtido pelo agente que executa uma ação. Portanto, se houver dois possíveis resultados A e B e $U_i(A) \geq U_i(B)$, o agente *i* prefere o que possuir maior utilidade (WOOLDRIDGE, 2009).

3.4 Consenso Distribuído

Processos falham, desastres naturais podem tirar data centers de operação em uma região. E, para isso, os engenheiros de confiabilidade dos sites precisam prever esses tipos de falhas e desenvolver estratégias para mantê-los operacionais, apesar de tais eventos indesejados. Essas estratégias geralmente implicam executar esses sistemas em ambientes distribuídos. Distribuir geograficamente um sistema é relativamente simples, mas também introduz a necessidade de manter uma visão consistente do estado do sistema, o que é uma empreitada mais difícil, com mais nuances (BEYER et al., 2016).

O consenso distribuído é eficaz para desenvolver sistemas confiáveis e altamente disponíveis, que exijam uma visão consistente de algum estado do sistema. Essa questão está relacionada à chegada de um acordo em um grupo de processos conectados por meio de uma rede de comunicação não confiável (BEYER et al., 2016). Este é um dos conceitos básicos utilizados no processamento distribuído, e também é utilizado nesta pesquisa para a comunicação e tomada de decisão entre os agentes.

Uma das estratégias usadas para tomar uma decisão conjunta em um cenário multiagente é a votação de maioria simples (*Simple majority voting*), em que a decisão final é tomada com base na função utilidade, o valor que trará mais benefícios para todos os agentes.

A votação por maioria simples é a forma mais simples de construir uma função de atualização. Os agentes mudam para uma estratégia alternativa se, até então, eles tiverem observado a adoção dessa outra estratégia por parte de outros agentes. Se mais de uma estratégia tiver sido observada, o agente escolherá aquela que tiver sido observada mais frequentemente (WOOLDRIDGE, 2009).

Esta estratégia é adotada neste trabalho para decidir onde os serviços do data center defeituoso serão restaurados. No nosso contexto, a função de utilidade é representada pelo *throughput*, a métrica que tentamos maximizar.

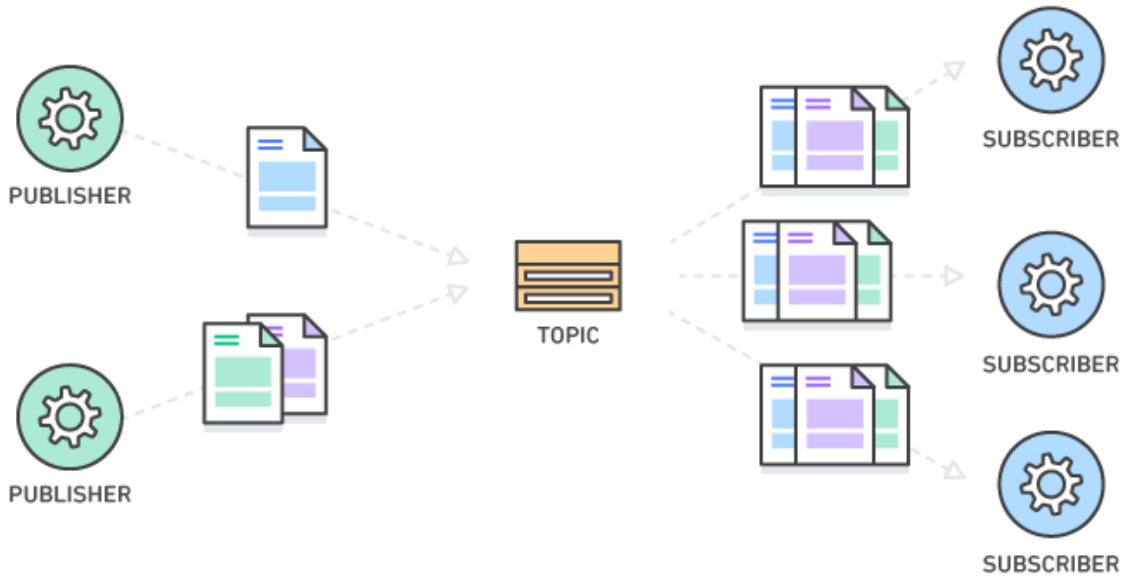
3.5 Sistema de mensagem *Publish-Subscribe*

É um modelo de comunicação baseado no compartilhamento de mensagem de forma assíncrona, conhecido como *sistemas baseados em eventos distribuídos*. É um sistema em que os *publishers* divulgam eventos (mensagens) e os *subscribers* manifestam interesse em recebê-los (COULOURIS et al., 2013).

O principal conceito em um sistema de mensagem *publish-subscribe* é o tópico em que os múltiplos *publishers* podem enviar mensagens para um tópico e todos os *subscribers* daquele tópico recebem todas as mensagens enviadas para esse tópico. O

modelo é extremamente útil quando um grupo de aplicações precisa ser notificado da ocorrência de um evento em particular (SANTOS JUNIOR, 2007). A Figura 4 apresenta uma visão geral do funcionamento desse sistema (AWS, 2018).

Figura 4 – Sistemas *Publish-Subscribe*



Fonte: AWS (2018)

A tarefa central desse sistema é combinar as assinaturas com os eventos publicados e garantir a entrega correta das notificações dos eventos. Os sistemas *publish-subscribe* são utilizados em diversos domínios de aplicações, particularmente aos relacionados à disseminação de eventos em grande escala. A saber:

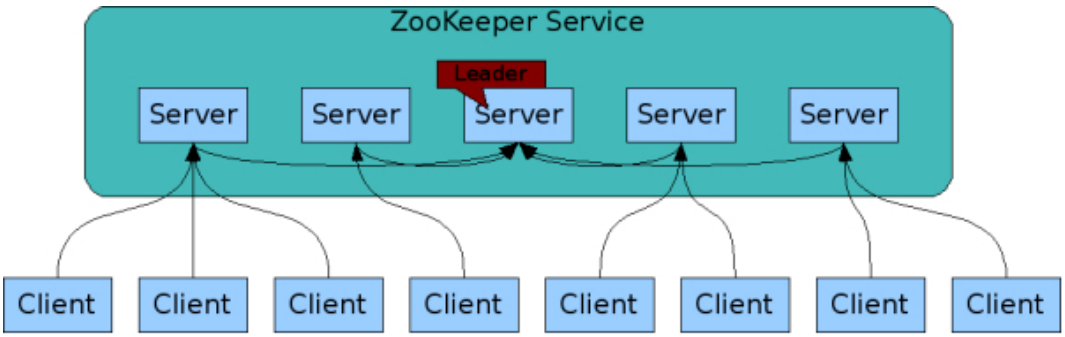
- sistema de informação financeira;
- áreas com divulgação ao vivo de dados em tempo real (incluindo feeds RSS);
- suporte para trabalho cooperativo (em que vários participantes precisam ser informados sobre eventos de interesse compartilhado);
- suporte para computação ubíqua, incluindo o gerenciamento de eventos provenientes de infraestrutura ubíqua (por exemplo, eventos de localização) e
- um grande conjunto de aplicativos de monitoramento, incluindo monitoramento de rede na Internet (COULOURIS et al., 2013) .

3.6 Apache Zookeeper

De acordo com Foundation (2018b), o Apache Zookeeper é um serviço de coordenação distribuído e de código aberto para aplicações distribuídas. Ele expõe um conjunto simples de primitivas sobre as quais as aplicações distribuídas podem ser construídas para implementar serviços de alto nível para sincronização, manutenção de configuração e grupos. Dessa forma, ele permite que processos distribuídos coordenem-se uns aos outros através de um espaço de nomes hierárquico compartilhado que é organizado de forma similar a um sistema de arquivos padrão.

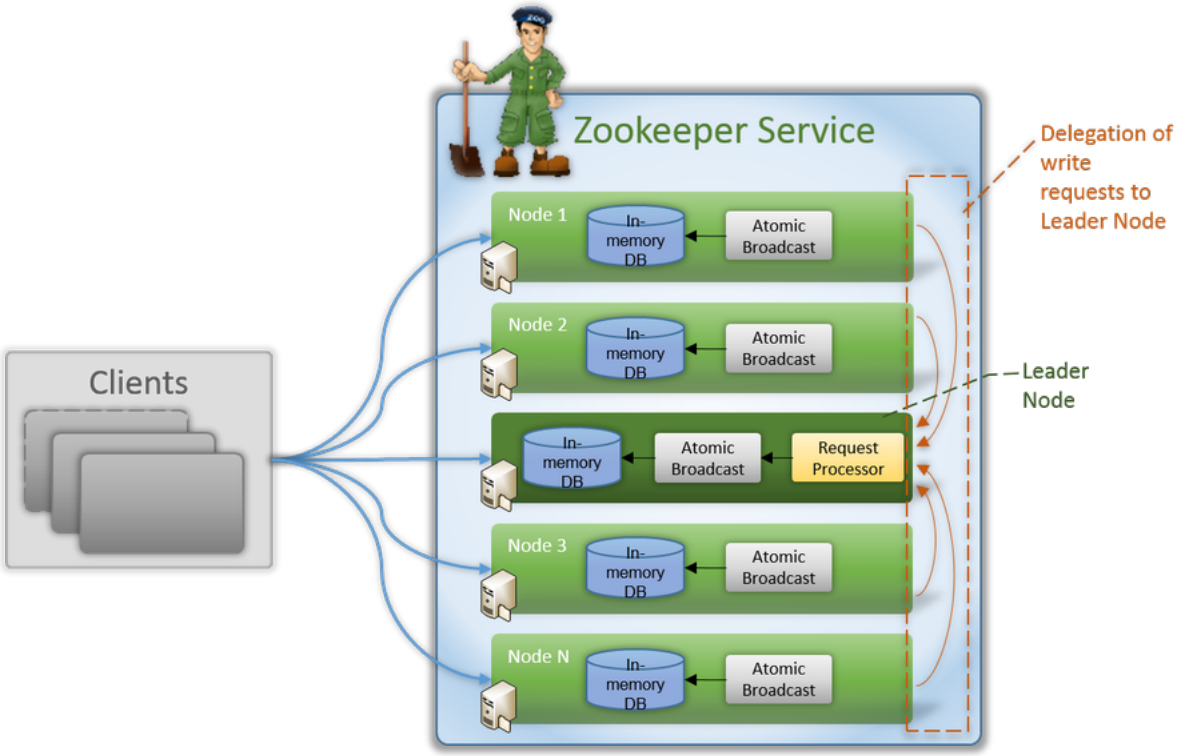
Além das características acima mencionadas, o Zookeeper também permite a duplicação do próprio servidor, tornando-o tolerante a falhas. Na Figura 5, será mostrada a arquitetura do Zookeeper.

Figura 5 – Arquitetura do Zookeeper



Ao detalhar-se mais a arquitetura do Zookeeper, pode-se entender o seu funcionamento interno, que é mostrado na Figura 6 e descrito a seguir.

Figura 6 – Arquitetura do serviço do Zookeeper



De acordo com a Figura 6, é possível perceber os seguintes elementos:

1) ***Leader Node*:** o Nó líder é o único responsável por processar as requisições de escrita. Todos os outros Nós, chamados de seguidores, simplesmente delegam as chamadas de escrita dos clientes para o Nó líder.

2) ***Follower Nodes:*** todos os outros Nós, exceto o Nó líder, são chamados de nós seguidores. Um nó seguidor é capaz de servir a requisições de leitura por si só. Os nós seguidores também possuem um importante papel na eleição de um novo Nó líder caso o atual venha a se tornar indisponível.

Os componentes do Nó, mostrados na Figura 6, são os seguintes:

1) ***Request Processor:*** este componente está ativo apenas no nó líder e é responsável por processar as operações de escrita do cliente ou dos Nós seguidores. Uma vez que ele tenha processado a requisição de escrita, este componente se encarrega de difundir a informação para os Nós seguidores de tal modo que os mesmos possam ser atualizados apropriadamente.

2) ***Atomic Broadcast:*** este componente está presente tanto no Nó líder quanto nos Nós seguidores. Ele é responsável por difundir as mudanças para os outros

Nós, quando está executando no Nó líder, assim como receber as notificações de mudança, quando está presente nos Nós seguidores.

3) ***In-memory database (Replicated Database):*** este banco de dados em memória e replicado é responsável por armazenar os dados no Zookeeper. Cada Nó contém seu próprio banco de dados, o que permite a eles servir as requisições de leitura. Além disso, os dados também são escritos em sistema de arquivos, possibilitando a recuperação em caso de problemas com o *cluster.* Em caso de solicitações de escrita, o banco de dados em memória é atualizado apenas após elas terem sido escritas com sucesso no sistema de arquivos.

3.7 Apache Kafka

De acordo com Foundation (2018a), o Apache Kafka é uma plataforma de *streamming* distribuída que usa o Apache Zookeeper para gerenciar as *streams*. Como uma plataforma de *streamming* distribuída ela possui três características:

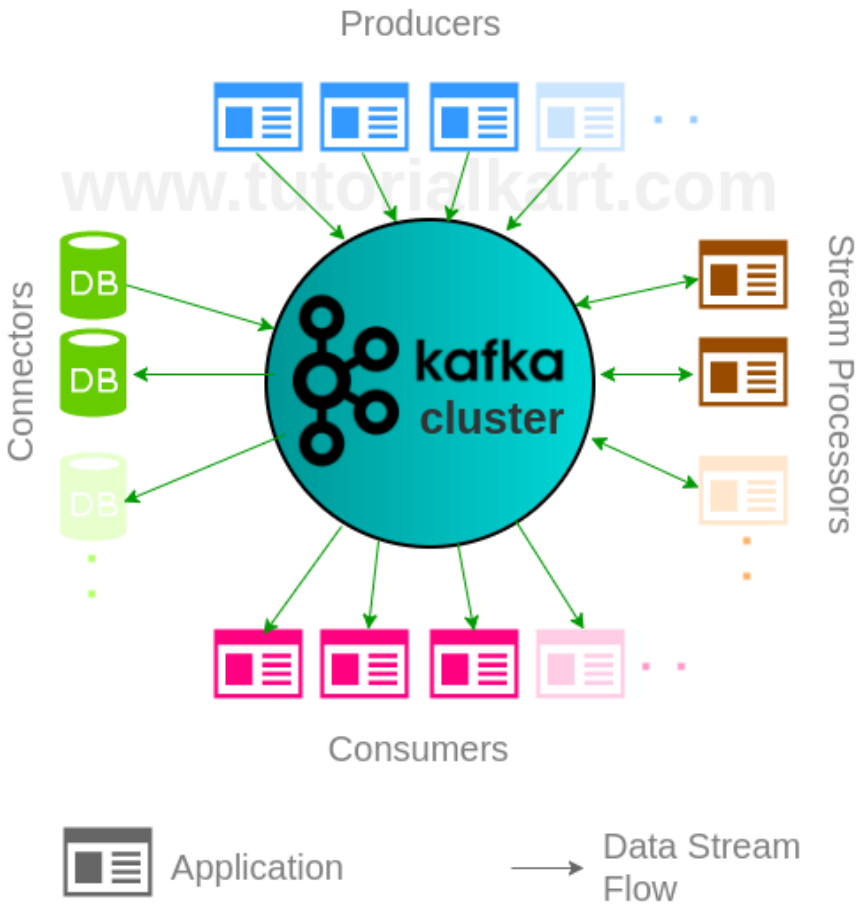
- o mecanismo *publish and subscribe* para *streams* de registros, semelhante ao mecanismo de fila de mensagens ou ao sistema *Enterprise messaging* usadas em java
- armazenamento de *streams* de registros em uma forma tolerante a falhas e
- processamento de *streams* de registro assim que elas ocorrem.

Os conceitos chaves utilizados pelo Apache Kafka são:

- o Kafka, que executa em um *cluster* composto por um ou mais servidores
- o *cluster*, que armazena *streams* de registros em categorias chamadas de tópicos e
- cada registro pode ser composto por chave, valor e um registro de tempo.

As principais **APIs** presentes no Apache Kafka são mostradas na Figura 7 e descritas a seguir.

Figura 7 – APIs do Apache Kafka



A seguir será descrito o passo a passo do ciclo de vida de uma nova entrada no *cluster do Kafka.*

1) Um registro é criado por um produtor e escrito em um dos **tópicos** existentes no *cluster* Kafka ou em um novo **tópico** que é criado para escrevê-lo.

2) Em seguida, o registro no **tópico** aguarda por um **Consumidor** ou um **Processador de *Stream***.

3) O **Processador de *Stream*** transforma o registro recém-criado em um novo registro, a fim de complementá-lo e escrevê-lo de volta no *cluster* em um novo **tópico**.

4) Pode haver múltiplas transformações no registro por múltiplos **Processadores de *Streams***.

5) Após todo esse processo, um **Consumidor** que tenha se inscrito para receber atualizações deste novo **tópico,** enfim pode consumi-lo e marcar o registro como consumido no banco de dados local.

6) Ao longo dessa jornada, as múltiplas transformações que ocorreram com o registro são registradas em bancos de dados relacionais usando **Conectadores.**

Qualquer aplicação pode tornar-se um **Produtor,** um **Consumidor**, ou um **Processador de *Streams*** baseado no papel que ela cumpre no *cluster* do Apache Kafka.

Após esse levantamento dos principais conceitos envolvidos neste trabalho, construiu-se uma metodologia para esta pesquisa, cuja principal contribuição é o desenvolvimento de um método baseado em agentes distribuídos, capaz de analisar condições de rede e identificar a vazão dos links que interligam um conjunto de data centers geograficamente distribuídos, em tempo real e, por fim, dar suporte à tomada de decisão para recuperação de serviços.

4 SOLUÇÃO PROPOSTA

Neste capítulo serão descritos os procedimentos metodológicos adotados nesta pesquisa. Inicialmente, é apresentada uma visão geral da metodologia e, em seguida, são apresentados os procedimentos para construção e implementação de cada módulo que compõe a solução para tomada de decisão.

4.1 Visão geral

Após realizar-se o levantamento dos estudos mais recentes que têm sido feitos pelos pesquisadores da área de *Live Migration* em redes, encontrar as lacunas nos estudos sobre tomada de decisão usando agentes distribuídos para a realização de tal atividade, conforme foi descrito no Capítulo 2 e verificar as oportunidades de melhoria discutidas na Seção 1.1, resolveu-se empreender uma pesquisa experimental. De acordo com Wazlawick (2014), o estilo de pesquisa “Apresentação de algo diferente”, consiste na apresentação de uma forma diferente de resolver um problema e é discutido como uma simples comparação entre técnicas. Optamos por esse tipo de pesquisa, pois não teríamos acesso às infraestruturas utilizadas nas pesquisas anteriormente realizadas na área para executar a nossa abordagem e comparar os resultados obtidos.

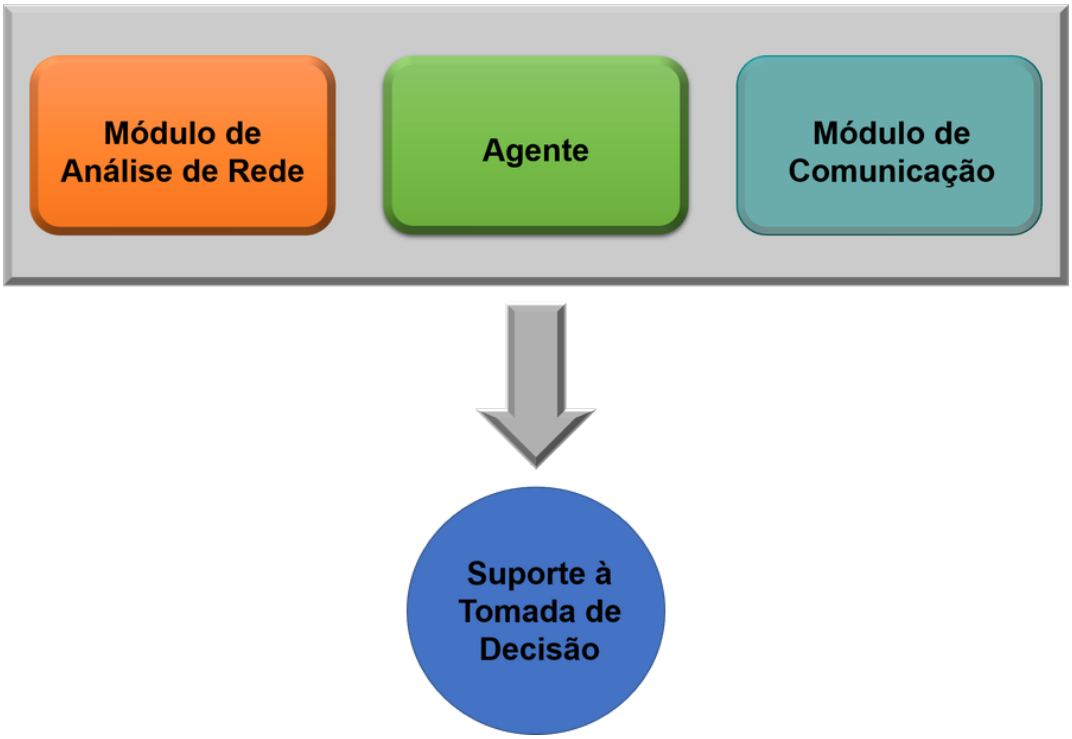
Dessa forma, este trabalho constitui-se como uma pesquisa experimental e**,** como tal, necessita-se buscar na literatura referente à metodologia científica os procedimentos e os métodos que podem ser utilizados a fim de construir uma pesquisa desse tipo. De acordo com Gil (2002), para se delinear uma pesquisa experimental, é necessário desenvolver uma série de passos que podem ser arrolados da seguinte forma:

1) Formulação do problema
2) Construção das hipóteses
3) Operacionalização das variáveis
4) Definição do plano experimental
5) Determinação dos sujeitos
6) Determinação do ambiente
7) Coleta de dados
8) Análise e interpretação dos dados

9) Apresentação das conclusões

A solução proposta, para dar suporte à tomada de decisão dos administradores de data center quanto à escolha do melhor site no que tange a vazão, é composta por três módulos: Módulo de rede, Agente e Módulo de comunicação. O primeiro é responsável por executar uma série de procedimentos para coleta de dados sobre a vazão da rede, que serão utilizados pelos Agentes. Este por sua vez, analisará os dados de modo a gerar um ranking com a vazão de todos os data centers e encaminhará para o Módulo de comunicação, responsável por compartilhar as informações de vazão com os demais agentes localizados em cada data center. Conforme pode-se observar na Figura 8 , a mesma apresenta uma visão geral da solução proposta:

Figura 8 – Arquitetura da Solução Proposta



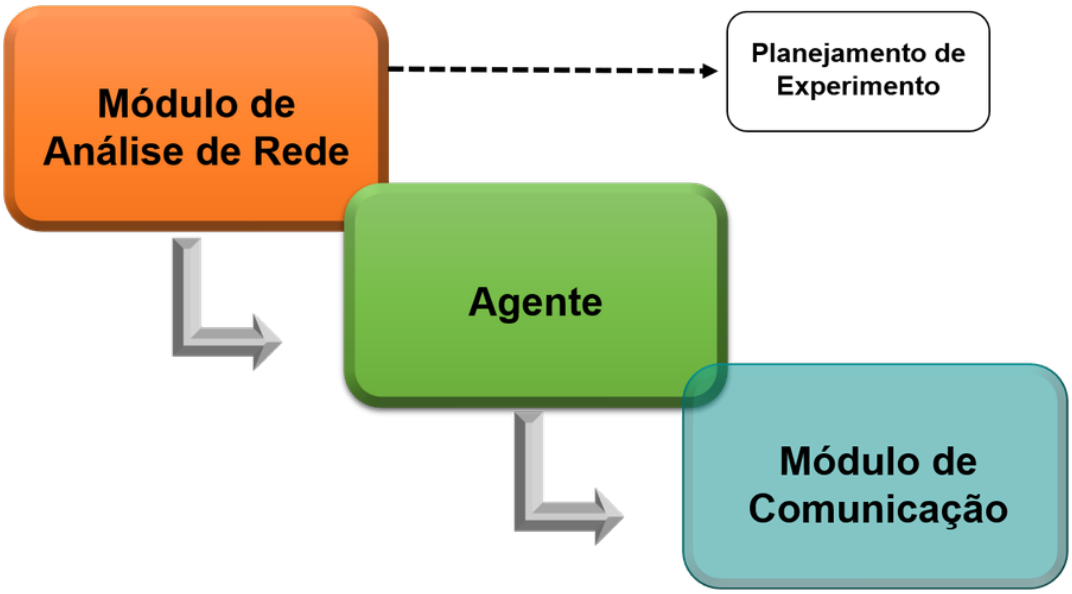
Na próxima seção discutiremos com mais detalhes a implementação dos três módulos que compõem a solução de suporte à tomada de decisão, o qual foi construído durante a realização desta pesquisa.

4.2 Procedimentos Metodológicos

Para a elaboração dos procedimentos metodológicos necessários para atingir os objetivos previamente estabelecidos, viu-se a necessidade de dividir-se a construção

da solução de acordo com cada um dos objetivos específicos e, com isso, criaram-se procedimentos metodológicos para atingir a cada um deles. A Figura 9 apresenta as etapas gerais da metodologia adotada e as seções seguintes descrevem de forma detalhada cada uma dessas etapas.

Figura 9 – Metodologia geral



4.2.1 Planejamento de Experimento

Ao coletar-se informações na literatura que trata da análise de desempenho de sistemas e de estatística, chega-se à conclusão que uma das estratégias adotadas para estudar a influência de variáveis em um sistema é utilizar o Planejamento de Experimentos, conforme descrito na Seção 3.2, ou a Análise de Variância que é uma das estratégias usadas de forma subjacente pela técnica de Planejamento de Experimentos.

Dessa forma, decidiu-se adotar essa estratégia para avaliar a influência de cada uma das métricas de rede estudadas (Delay, Jitter e Largura de Banda) sobre a variável resposta (Throughput). A fim de otimizar-se o tempo necessário para a realização dos experimentos, decidiu-se adotar três níveis apenas para cada uma das variáveis em estudo e com isso chegou-se a um experimento fatorial completo 3^3.

Para isso, coletou-se informações de diferentes condições de rede para definir os cenários necessários do Planejamento de Experimentos, medindo o *Round Trip Time* (RTT) de sete sites durante 4 dias (de sábado a quarta-feira) a partir de um servidor implantado no serviço ***Cloud At Cost*** localizado nos Estados Unidos. A seguir, elencamos os sites escolhidos.

1) Amazon CA (Canadá)
2) Amazon UK (Reino Unido)
3) Google (EUA)
4) Adidas ZA (Africa do Sul)
5) Loot ZA (Africa do Sul)
6) UOL (Brasil)
7) Oi (Brasil)

Após a coleta dos dados do RTT, usamos as relações apresentadas nas Equações 3.1 e 3.2 para calcular os respectivos valores de delay e jitter de cada amostra de RTT. A partir dos valores obtidos, do delay e jitter, de cada amostra para cada site, obtivemos o valor médio de atraso e jitter para cada site usando as relações mostradas na Equação 4.1. Os cenários que foram obtidos após esses procedimentos são mostrados na Tabela 4.

Tabela 4 – Cenários utilizados durante experimentos

Cenário	Delay(ms)	Jitter(ms)
1	12.59	2.06
2	11.35	1.69
3	4.99	0.59
4	8.057	2.056
5	14.23	1.37
6	80.44	1.40
7	13.26	1.61

Para replicar as condições mostradas na Tabela 4 em um cenário controlado (local), foi utilizado o **Netem** (FOUNDATION, 2018d), que é uma ferramenta que emula condições de rede. Ela pode ser usada para manipular delay, jitter e muitos outros fatores de rede. Para manipular a largura de banda durante nossos experimentos, usamos o **Token Bucket Filter** (FOUNDATION, 2018f), que é outra ferramenta da Linux Foundation que pode ser usada em conjunto com o Netem. Para analisar os efeitos de cada fator (delay, jitter e largura de banda) ao longo do tempo para transferir

um volume de dados, decidimos fazer um planejamento de experimento usando os cenários 2, 3 e 6.

Então, por decisão de projeto, optou-se por fazer um planejamento fatorial completo usando os valores de delay e jitter dos cenários escolhidos. Para medir o impacto da largura de banda, usamos os valores: 1 Gbit, 750 Mbit e 500 Mbit. Após decidir os níveis dos fatores, configuramos os cenários de análise com todas as combinações de valores dos fatores, resultando em um design fatorial 3^3 com três repetições resultando em 81 experimentos. Os resultados da Análise de Variância são mostrados na Figura 10 .

Figura 10 – Resultados do ANOVA

Análise de Variância

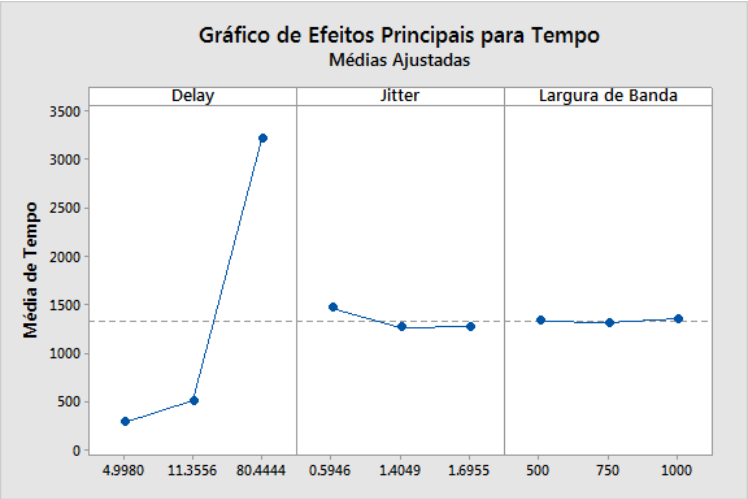
Fonte	GL	SQ (Aj.)	QM (Aj.)	Valor F	Valor-P
Modelo	26	145649862	5601918	716.94	0.000
Linear	6	143941859	23990310	3070.30	0.000
Delay	2	143200888	71600444	9163.49	0.000
Jitter	2	716843	358421	45.87	0.000
Bandwidth	2	24129	12064	1.54	0.223
Interações de 2 fatores	12	1670643	139220	17.82	0.000
Delay*Jitter	4	1604687	401172	51.34	0.000
Delay*Bandwidth	4	46997	11749	1.50	0.214
Jitter*Bandwidth	4	18958	4740	0.61	0.660
Interações de 3 fatores	8	37360	4670	0.60	0.776
Delay*Jitter*Bandwidth	8	37360	4670	0.60	0.776
Erro	54	421938	7814		
Total	80	146071800			

Como pode-se observar na Figura 10, a variância do tempo de transferência (variável resposta do experimento) pode ser atribuída ao Delay e o Jitter, o que é demonstrado pelos valores de *p-value* menor que 5%. Também percebe-se pelos resultados obtidos que o produto entre essas métricas ($Delay \times Jitter$) impacta na vazão final do link.

Para analisar os efeitos provocados por cada um dos parâmetros de rede sobre a vazão, optou-se por utilizar o gráfico de efeitos principais, mostrado na Figura 11. De acordo com Minitab (2018b), para analisar esse gráfico se faz necessário avaliar dois padrões:

- Quando a linha é horizontal (paralela ao eixo x), não há nenhum efeito principal e
- Quando a linha não é horizontal, existe um efeito principal. Quanto maior a inclinação da linha, maior é a magnitude do efeito principal.

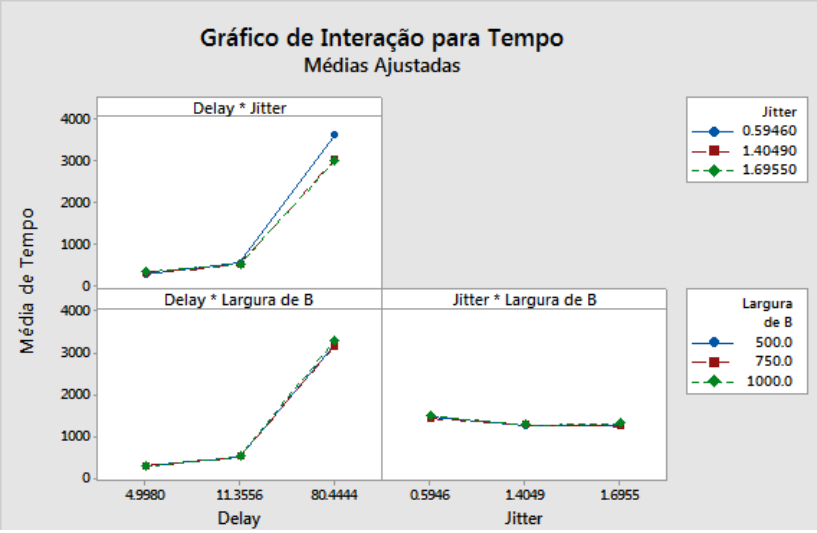
Figura 11 – Efeitos Principais dos parâmetros estudados



De acordo com o gráfico mostrado na Figura 11, pode-se perceber que o parâmetro **Largura de Banda** não produz efeito principal significativo sobre o Tempo de Transferência e, consequentemente, sobre a vazão da rede. Por outro lado, pode-se observar que os parâmetros **Delay** e **Jitter** possuem um efeito sobre o Tempo de Transferência. O primeiro tem um efeito principal de grande magnitude sobre o Tempo de Transferência e o segundo de menor magnitude sobre a variável resposta do experimento.

Além dos efeitos principais, decidiu-se obter o Gráfico de Interação que, de acordo com Minitab (2018b), é útil para visualizar as interações entre os fatores em análise (Delay, Jitter, Largura de Banda).

Figura 12 – Interações dos parâmetros estudados



Os resultados mostrados por um gráfico de interação, de acordo com Minitab (2018a), devem ser analisados da seguinte forma:

- **Linhas paralelas:** não ocorre nenhuma interação
- **Linhas não paralelas:** ocorre uma interação, quanto mais não paralelas são as linhas, maior é a força da interação.

Dessa forma, ao observarmos o Gráfico de Interação para o Tempo de Transferência, percebemos que os fatores não possuem interação entre si, uma vez que se sobrepõem. Esses resultados corroboram com a ideia defendida por (FOROUZAN, 2007) de que a relação mais importante para medir o impacto do **Delay** e da **Largura de Banda** sobre o **Tempo de Transmissão** é o produto entre as duas primeiras.

De posse dessas informações, optou-se por utilizar principalmente as métricas de Delay e Jitter, devido ao seu maior impacto sobre o Tempo de Transmissão e, consequentemente, sobre a Vazão, para a construção do módulo de análise de rede proposto neste trabalho. Na seção a seguir, descreveremos como foi implementado este módulo.

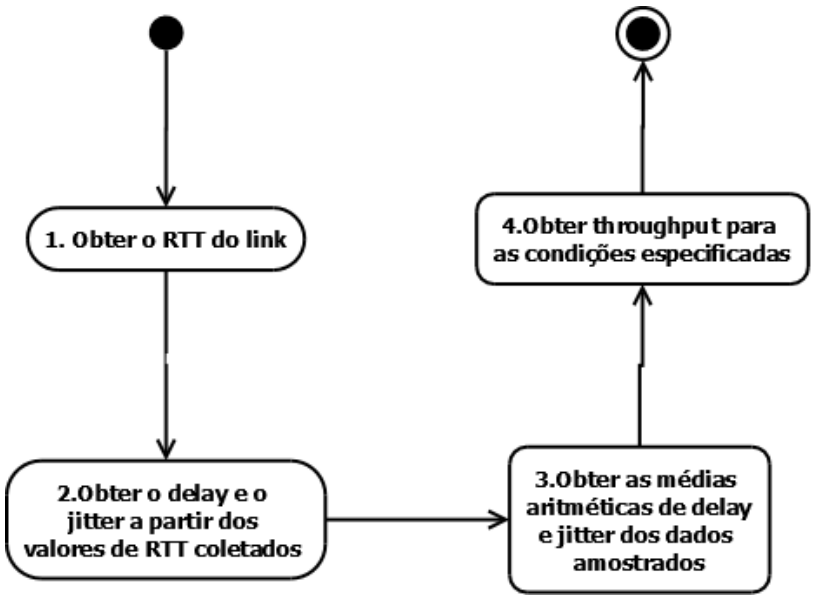
Com isso, tem-se o procedimento necessário para cumprir com o objetivo de **Utilizar métodos experimentais para estudar o impacto de parâmetros de rede sobre a vazão** e, de posse dos parâmetros que têm maior influência sobre o Throughput, faz-se necessário construir os procedimentos a fim de realizar a análise da rede

4.2.2 Módulo de Análise de Rede

Após seguir os procedimentos descritos na Seção anterior, chegou-se à conclusão de que dentre os parâmetros de rede elencados, Delay, Jitter e Largura de Banda, os que possuem maior influência sobre o Throughput são os dois primeiros.

A partir dessa conclusão, foi possível construir os procedimentos metodológicos necessários para cumprir o objetivo de **Definir um procedimento de análise das condições de rede**, conforme mostrado na Figura 13. A seguir, descreveremos cada um dos passos da metodologia de análise de rede.

Figura 13 – Metodologia de Medição



1) **Obter o RTT do link:** como a maior parte das ferramentas disponíveis não fornece o delay e o jitter de um link diretamente, optou-se por obter o RTT (Round Trip Time) e realizar as transformações necessárias para obter as métricas de interesse a partir do RTT. Desse modo, nesse passo realizam-se requisições ao servidor de interesse e obtém-se o Round Trip Time de cada requisição feita.

2) **Obter o delay e o jitter a partir dos valores de RTT coletados:** após obter os valores do RTT do servidor de interesse, derivam-se as métricas de delay e jitter a partir das relações mostradas nas Equações 3.1 e 3.2.

3) **Obter as médias aritméticas de delay e jitter dos dados amostrados:** após transformarmos as diversas amostras de RTT coletadas no primeiro passo em amostras de delay e jitter usando as relações mostradas nas Equações 3.1 e 3.2,

computam-se as médias aritméticas das amostras de delay e jitter obtidas no passo anterior usando as relações mostradas na Equação 4.1. Os significados dos termos D_i e J_i estão de acordo com a Tabela 3.

$$\overline{D} = \sum_{i=1}^{N} \frac{D_i}{N} \qquad \overline{J} = \sum_{i=1}^{N} \frac{J_i}{N} \tag{4.1}$$

4) **Obter o throughput para as condições especificadas:** após obter as médias aritméticas das amostras de delay e jitter no passo anterior, tem-se o subsídio necessário para caracterizar uma determinada condição de rede, uma vez que, como discutido na seção anterior, esses são os parâmetros de rede que mais impactam o Throughput. De posse dessas informações, um delay e um jitter representativos para um dado link, obtemos o throughput do link em estudo e, com isso, completa-se a análise do link,visto que se tem as informações suficientes a respeito das condições de rede para elaborar-se uma decisão.

Para implementar os procedimentos metodológicos do mecanismo de análise, fez-se um estudo preliminar a fim de encontrar ferramentas que pudessem obter as métricas de interesse (Delay, Jitter, Throughput).

Durante esse estudo, foram obtidas diversas ferramentas existentes nativamente em sistemas operacionais Linux que são capazes de mensurar o Round Trip Time em um *link* existente entre dois lugares. Aqui, destacam-se duas ferramentas:

1) **Ping:** ferramenta tradicional na análise de redes que utiliza o protocolo ICMP (Internet Control Mensage Protocol) para mensurar o Round Trip Time entre hosts através do envio de **ICMP ECHO_REQUEST** (FOUNDATION, 2018e) e

2) **Hping3:** ferramenta utilizada para mensurar o Round Trip Time de um link através do envio de diferentes tipos de pacote TCP/IP (FOUNDATION, 2018c).

Devido ao fato de a ferramenta **Hping3** ser mais robusta quanto à personalização do tipo de requisição TCP, inclusive em relação à possibilidade de personalizar a porta em que o teste é realizado, e ao fato de a ferramenta **Ping** estar disponível nativamente tanto em sistemas operacionais tanto Windows quanto Linux, optou-se por tornar flexível, dentro do módulo de medição, a utilização da ferramenta para a obtenção do Round Trip Time.

Outra decisão bastante importante para a implantação da solução era a definição da infraestrutura na qual seriam implantados os agentes a fim de testá-los e gerar os resultados da pesquisa. Dessa forma, tentou-se criar um ambiente experimental todo em nuvem (data centers reais). Porém, encontramos dificuldades em mensurar a

vazão real dos *links*, pois os provedores em geral não disponibilizam essa informação. Até houve uma tentativa frustrada de transferir uma massa de dados de 30 GB entre dois data centers geograficamente distribuídos para mensurar o *throughput*, pois o provedor bloqueou o nosso acesso.

A fim de tornar o mecanismo desenvolvido durante a realização desta pesquisa flexível, buscou-se, durante o processo de desenvolvimento, criar interfaces de modo que, posteriormente, pudéssemos realizar testes em nuvem. Assim, foi definido um cenário de experimentação em infraestrutura local, em que a proposta foi emular condições de rede da Internet, e outro cenário de experimentação, embora mais limitado, em nuvem. A abordagem de testes locais nos proporcionou maior controle sobre o ambiente de experimentação, o que possibilitou a medição da vazão com condições de rede emuladas com parâmetros obtidos na WAN.

Outra abordagem utilizada durante a implementação do projeto fez uso da biblioteca **Speedtest.net Python script** que é capaz de mensurar o RTT, a velocidade de download e a velocidade de Upload, ou seja, analisa a vazão do link. Dessa forma, o mecanismo construído durante esta pesquisa torna-se flexível o suficiente para que a infraestrutura de medição possa ser implantada localmente ou através da utilização do serviço **SpeedTest.net**, que já é disponibilizado comumente para *desktops* e *smartphones*.

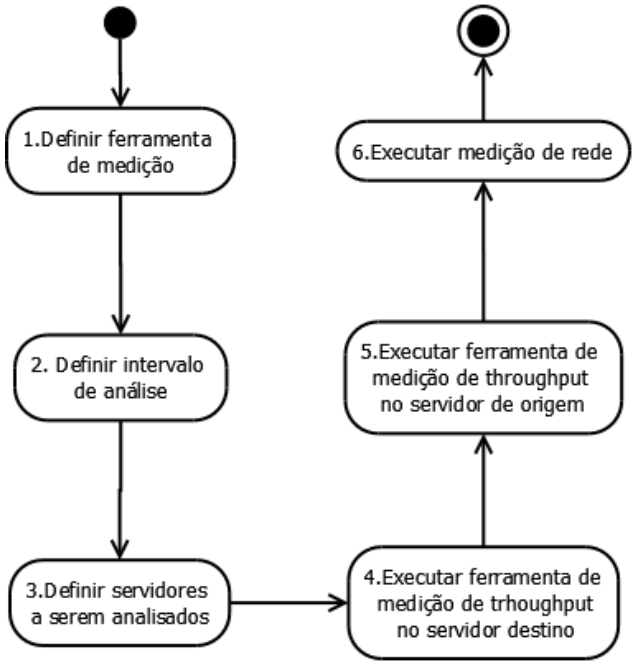
Com isso, nas próximas seções descreveremos a implementação do mecanismo de medição utilizando a infraestrutura local e em nuvem, respectivamente.

A) Implantação em Infraestrutura Local

Para implementar o mecanismo de medição em infraestrutura local foi construído um procedimento a ser seguido pelo administrador de sistema a fim de, ao final, ter-se uma infraestrutura de medição mais adequada às condições existentes dento do Data Center em que o mecanismo estiver sendo usado.

A seguir, será descrito cada um dos passos a serem cumpridos pelo administrador de sistema, a fim de construir a infraestrutura necessária à implantação do mecanismo de medição. A Figura14 apresenta o diagrama de atividades do procedimento de implantação.

Figura 14 – Procedimento de implantação de Infraestrutura de Medição Local



1) **Definir ferramenta de medição:** a primeira decisão a ser feita pelo administrador de sistema, para a realização da medição de condições de rede, é a definição da ferramenta a ser usada para obter amostras do RTT (*Round Trip Time*).

2) **Definir Intervalo de análise:** após a definição da ferramenta a ser usada para medir o RTT, o administrador de sistema deve definir o intervalo de tempo entre medições de rede. Dessa forma, neste passo, o administrador irá definir o tempo que o agente utilizará para realizar novas medições das condições de rede.

3) **Definir servidores a serem analisados:** neste passo, o administrador de sistema definirá os endereços dos servidores que o agente, implantado no data center, irá ter contato para obter informação a respeito das condições de rede do link.

4) **Executar ferramenta de medição de throughput no servidor de destino:** neste passo, o administrador de sistema irá executar o **Iperf** (ferramenta de medição de throughput) no modo **servidor** no Data Center de destino das medições.

5) **Executar ferramenta de medição de throughput no servidor de origem:** neste passo, o administrador de sistema irá executar o **Iperf** (ferramenta de medição de throughput) no modo **cliente** no Data Center de origem das medições.

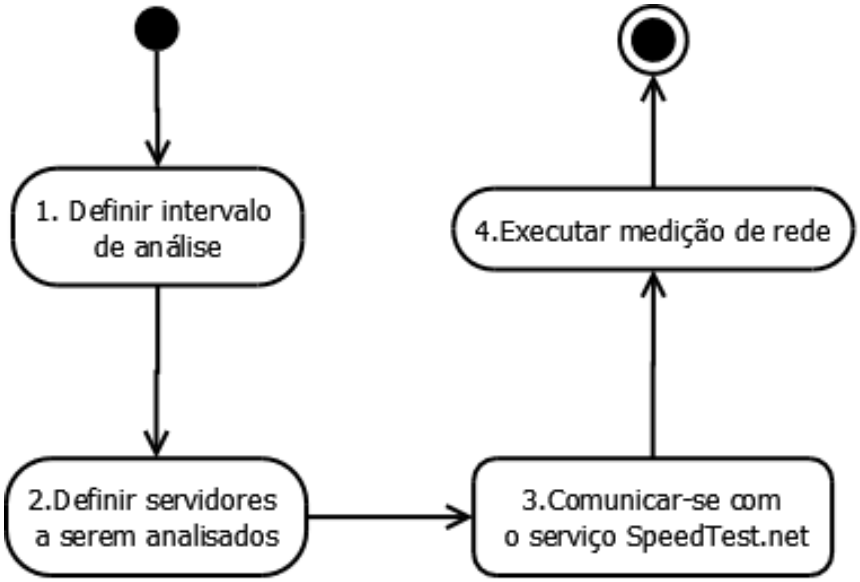
6) **Executar medição de rede:** ao completar os passos anteriores, a infraestrutura de medição necessária para que o mecanismo desenvolvido nesta pesquisa possa funcionar estará montada.

Os procedimentos descritos nesta seção definem uma das formas de construir a infraestrutura necessária para implantar o módulo de medição de rede em uma arquitetura local. Na próxima seção, serão descritos os procedimentos para implantar o mecanismo de medição utilizando uma infraestrutura em nuvem.

B) Implantação em Nuvem

Nesta seção, serão descritos os procedimentos a serem adotados pelo administrador de sistema caso ele opte por implantar o mecanismo de medição utilizando o serviço do **Speedtest.net**. A Figura15 mostra um diagrama de atividades que será descrito a seguir.

Figura 15 – Procedimento de implantação de Infraestrutura de Medição em Nuvem



1) **Definir intervalo de análise:** neste passo, o administrador de sistema irá definir o intervalo de tempo entre medições de rede. Dessa forma, será definido o tempo que o agente utilizará para realizar novas medições das condições de rede.

2) **Definir servidores a serem analisados:** neste passo, o administrador de sistema definirá os endereços dos servidores que o agente implantado no Data Center irá ter contato para obter informação a respeito das condições de rede do link.

3) **Comunicar-se com o serviço Speedtest.net:** para o funcionamento desta parte, durante a pesquisa, utilizou-se a biblioteca **pyspeedtest** a fim de estabelecer uma comunicação com o serviço **SpeedTest.net** e, com isso, obter informações sobre o **RTT**, a **taxa de download** e a **taxa de upload.**

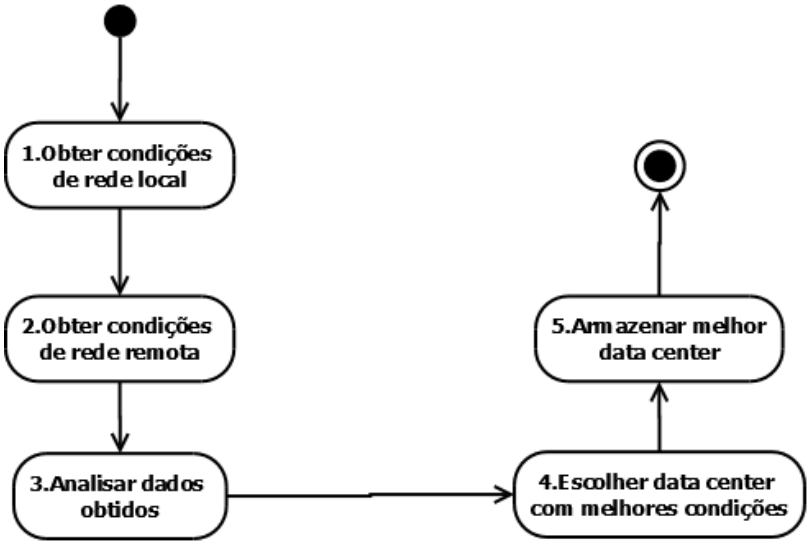
4) **Executar medição de rede:** ao completar os passos anteriores, a infraestrutura de medição em nuvem necessária para que o mecanismo desenvolvido nesta pesquisa possa funcionar estará montada.

Após completar os passos descritos anteriormente, tem-se o procedimento metodológico necessário para cumprir o objetivo de **Definir um procedimento de análise das condições de rede**, conforme estabelecido previamente. Na seção a seguir, serão definidos os agentes que serão implantados em cada Data Center de interesse.

4.2.3 Agente

Para cumprir o objetivo de construir um agente que fosse capaz de coletar as informações das condições de rede locais e remotas, decidiu-se elaborar o procedimento metodológico mostrado na Figura 16. A seguir, será descrito cada um dos passos necessários para a construção do agente.

Figura 16 – Metodologia para tomada de decisão



1) **Obter condições de rede local:** neste passo, o agente utiliza as ferramentas necessárias para realizar a análise de rede, conforme descrito na seção anterior.

Desse modo, ele utiliza ferramentas de rede convencionais para mensurar o RTT (Round Trip Time) do link e aplica as transformações necessárias, descritas na Seção 3.1, para, ao final, ter as informações das condições de rede dos links do Data Center no qual ele está implantado.

2) **Obter condições de rede remotas:** ao realizar o passo anterior, cada agente, implantado em cada um dos Data Centers que se comunicam, possui informações a respeito dos links locais. Dessa forma, utilizando bibliotecas de comunicação assícronas, cada um dos agentes se comunica com todos os outros agentes que estão implantados nos outros data centers, a fim de obter as informações coletadas por eles a respeito dos seus respectivos links locais.

3) **Analisar dados obtidos:** após estar de posse das informações a respeito das condições de rede locais e remotas, cada agente tem os subsídios necessários para tratar os dados obtidos a fim de construir um *ranking* dos Data Centers que possuem melhores condições de rede em relação ao Data Center no qual ele está implantado.

4) **Escolher data center com melhores condições**: após construir o *ranking* no passo anterior, cada um dos agentes terá os subsídios necessários para tomar a decisão a respeito do Data Center com melhores condições de rede em relação ao Data Center no qual ele está implantado e, com isso, conseguir informar, *a posteriori,*outros agentes a respeito do DC no qual devem ser restaurados os serviços do centro de dados em que ele está executando.

5) **Armazenar o melhor data center:** neste passo, cada agente irá armazenar as informações do data center escolhido no passo anterior em uma estrutura de dados na qual seja fácil transmitir a outros agentes a decisão tomada por cada um deles.

Para implementar os agentes para tomada de decisão precisava-se construir um algoritmo capaz de usar o mecanismo de análise de rede com uma das estratégias definidas anteriormente e utilizar um mecanismo de comunicação distribuída, como será descrito na próxima seção.

Para cumprir o passo **Obter condições de rede local** (Figura 16), utilizou-se o módulo de medição de rede com uma das estratégias definidas na seção anterior (Local ou Nuvem). O passo **Obter condições de rede remota** foi cumprido utilizando o mecanismo de comunicação distribuído que será explicado na próxima seção.

Para cumprir o passo **Analisar dados obtidos**, desenvolveu-se um algoritmo que utiliza uma estrutura de dados do tipo *Tabela Hash*. As tabelas *hash*, segundo Goodrich e Tamassia (2013), são uma forma eficiente de implementar o **TAD** (Tipo

Abstrato de Dados) do tipo Mapa, que permite localizar elementos rapidamente usando chaves.

Dessa forma, o algoritmo desenvolvido utiliza os dados obtidos para construir um ranking baseado no **TAD** mapa em que as chaves são os endereços dos servidores e o valor são as pontuações obtidas por esses servidores.

Para cumprir o passo **Escolher data center com melhores condições**, o algoritmo desenvolvido ordena as chaves (endereços dos servidores) do mapa construído no passo anterior pelos seus valores (pontuações obtidas pelos servidores).

Por fim, ao construir o *ranking* no passo anterior, o agente consegue estabelecer qual a colocação de cada Data Center em relação ao *Throughput*, cumprindo o passo **Armazenar o melhor data center.**

O algoritmo do agente é apresentado na Figura 17, podendo ser dividido em duas partes: 1) Decisão local e 2) Decisão distribuída.

Figura 17 – Algoritmo do agente para tomada de decisão

Algorithm 1 Agent's algorithm

```
1:  hosts ← hashTable
2:  rankedServers ← hashTable
3:  bestServers ← hashTable
4:  procedure GETSERVERSCONDITIONS(hostnameList)
5:      hostnames ← split(hostnameList,',')
6:      i ← 0
7:      while i < length(hostnames) do
8:          hostname ← hostnames[i]
9:          hosts[hostname] ← GetHostData(hostname)
10:         i ← i + 1
11:     hosts ← rankServers(hosts)
12: procedure GETHOSTDATA(hostname)
13:     hostData['TP'] ← getDwRate(hostname)
14:     hostData['RTT'] ← getRTT(hostname)
15:     return hostData
16: procedure RANKSERVERS(hosts)
        rankedServers ← orderDescByTP(hosts)
        return rankedServers
17: procedure SENDBESTSERVER
18:     bestServer ← rankedServers.pos[0]
19:     send(bestServer)
20: procedure RECVBESTSERVER(srvAddr, bestSrvData)
21:     bestServers[srvAddr] ← bestSrvData
22: procedure JOINTDECISION
23:     remove hosts[servAddrLost]
        jointBestServer ← hashTable
24:     remove rankedServers[srvAddrLost]
25:     hostBestServer ← rankedServers.pos[0].key
26:     jointBestServer[hostBestSever] ← 1
27:     i ← 0
28:     while i < length(hosts) do
29:         srvAddr ← hosts.pos[0].key
30:         bestSrvAddr ← bestServers[srvAddr]
31:         count ← jointBestServer[bestSrvAddr]
32:         jointBestServer[bestSrvAddr] ← count + 1
33:         i ← i + 1
34:     orderDesc(jointBestServer.rankCount)
        return jointBestServer.pos[0]
```

O procedimento **GetServersConditions** permite obter as condições de rede dos Data Centers com os quais o agente possui comunicação. Neste procedimento, é passada, como parâmetro, uma lista de endereços dos DCs dos quais o agente irá medir as condições de rede e armazenar os resultados obtidos na variável ***rankedServers***.

Para realizar a medição de rede, o agente utiliza o procedimento **GetHostData**, que recebe como parâmetro o endereço do Data Center com o qual se deseja medir o **RTT** e o **Throughput**. Os dados obtidos são colocados em uma estrutura de Mapa com chave igual ao nome do parâmetro e retornado para o procedimento **GetServersConditions**.

Ao final do procedimento **GetServersConditions**, realiza-se uma chamada ao procedimento **RankServers**, a fim de ordenar o Mapa ***hosts*** de acordo com o valor do throughput em ordem decrescente. Com isso, o agente conclui a parte da tomada de decisão local.

Para cumprir o objetivo de realizar uma tomada de decisão conjunta**,** o agente utiliza os procedimentos **SendBestServer** e **RecvBestServer**. O procedimento **SendBestServer** utiliza a variável ***rankedServers*** para enviar aos outros servidores o servidor com melhor desempenho. Já o procedimento **RecvBestServer** utiliza a variável bestServeres para armazenar as informações dos Data Centers com melhores condições de rede recebidas de outros agentes.

Ao final da execução dos procedimentos acima citados, o agente tem as condições suficientes para executar o procedimento **JointDecision**, que irá utilizar as informações coletadas local e remotamente para construir um *ranking* baseado nelas, atingindo, dessa forma, o objetivo da pesquisa.

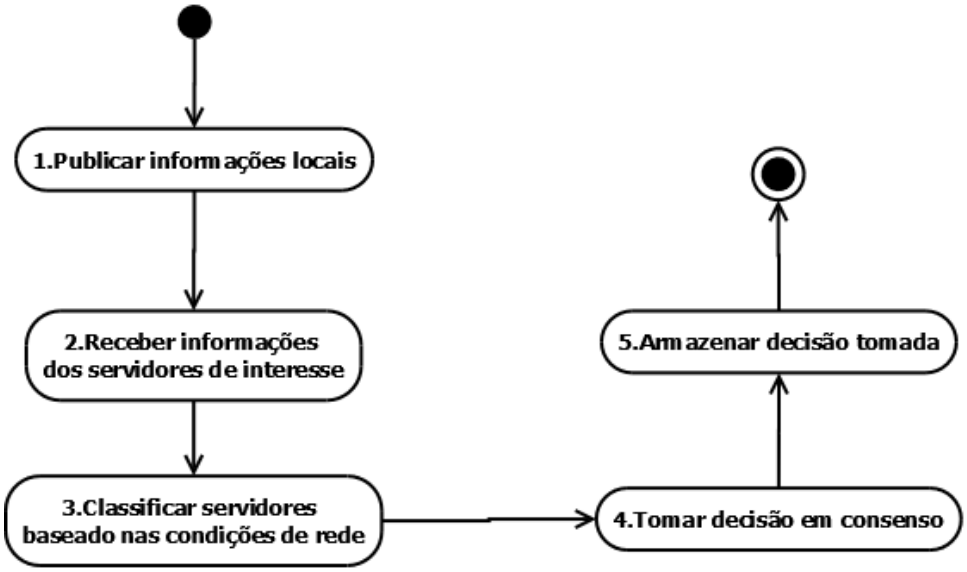
Após cumprir cada um dos passos estabelecidos anteriormente, cumpri-se o objetivo de **Construir agentes para auxiliar no processo de tomada de decisão** e, dessa forma, após definir um procedimento para comunicação, finalizar a construção do agente e, com isso, cumprir o objetivo de **Implantar uma arquitetura de comunicação entre os data centers geograficamente distribuídos**. Na seção a seguir, definimos os procedimentos metodológicos para criar um mecanismo de comunicação entre os agentes e, com isso, fazê-los tomar decisões baseadas em consenso distribuído.

4.2.4 Módulo de Comunicação

Para cumprir o objetivo de implantar uma arquitetura de comunicação que permita a troca de informações entre agentes**,** decidiu-se elaborar o procedimento

metodológico mostrado na Figura 18. A seguir será descrito cada um dos passos necessários para a construção do mecanismo.

Figura 18 – Metodologia para obtenção de consenso distribuído



1) **Publicar informações locais:** neste passo, o agente responsável por coletar as informações a respeito das condições de rede, de cada um dos links do data center no qual ele está hospedado, consolida as informações coletadas em uma estrutura de dados que possa ser enviada para um servidor responsável por enviar essas informações a todos os agentes que possuem interesse em recebê-la.

2) **Receber informações dos servidores de interesse:** neste passo, após enviar os dados coletados no passo anterior, o agente consulta o servidor de mensagens para obter atualizações das condições de rede dos Data Centers de interesse.

3) **Classificar servidores baseado nas condições de rede:** neste passo, após ter coletado as informações a respeito das condições de rede locais e remota, o agente tem os subsídios necessários para classificar os Data Centers com os quais mantém ligação baseado nas condições de rede deles com relação ao Data Center no qual ele está hospedado e em relação a cada um dos outros Data Centers com qual ele mantém comunicação.

4) **Tomar decisão em consenso:** com base nas classificações feitas por cada um dos agentes no passo anterior, pode-se chegar a um consenso entre eles usando o método da **votação por maioria simples**. Para usar esse método,

cada um dos agentes constrói um *ranking* local do *throughput* do link com cada um dos DCs com os quais ele possui ligação em ordem decrescente. Após a construção desse *ranking* local, cada um dos agentes envolvidos na comunicação compartilha os *rankings* previamente construídos a fim de possibilitar a construção de um *ranking* global. Desse modo, ao final desse procedimento, tem-se uma lista indicando para cada Data Center qual o melhor DC para restaurar o serviço dele.

5) **Armazenar decisão tomada:** com base no *ranking* construído no passo anterior, cada um dos agentes pode guardar a decisão tomada de modo que ao perder contato com um determinado Data Center os serviços do mesmo possam ser restaurados no melhor DC baseado no *ranking*.

Ao final da realização deste procedimento metodológico para construção de um mecanismo de comunicação, tem-se uma solução capaz de aferir as condições de rede, reunir as informações mensuradas, realizar a troca de informações entre agentes e, com isso, construir um *ranking* que informe a todos os agentes que fazem parte de um grupo de Data Centers qual o melhor DC para restaurar os serviços de um Centro de Dados que venha a falhar.

Os procedimentos de Análise de Rede, Tomada de Decisão local e Troca de informações entre agentes devem ser feitos periodicamente com uma frequência a ser definida pelos administradores dos Data Centers, de modo que se tenham informações atuais e com isso se possa tomar a decisão de restaurar os serviços de um DC falho no menor tempo possível, minimizando, dessa forma, o *downtime* e aumentando a disponibilidade dos serviços em questão.

Para implementar uma arquitetura de comunicação distribuída, considerando as necessidades explicitadas durante a realização da pesquisa, necessitava-se de uma infraestrutura de comunicação que

- permitisse a comunicação entre os agentes de forma confiável, uma vez que não se podia considerar perdas por indisponibilidade do serviço de troca de mensagens;
- possuísse um serviço de coordenação entre os agentes de modo a possibilitar a troca de mensagens sem que houvesse a possibilidade de perda de mensagens ou duplicidade das mesmas e
- permitisse a redundância do serviço e o armazenamento das mensagens trocadas entre os agentes, de modo que, em caso de falha de um Data Center, o agente hospedado no Data Center falho pudesse obter as mensagens de seu

interesse trocadas anteriormente pelos agentes hospedados em Data Centers funcionais.

Com essas necessidades em mente, empreendeu-se uma busca por ferramentas que atendesse a esses requisitos de pesquisa a fim de não apenas poupar tempo para a produção mas também para teste e validação das funcionalidades requeridas, caso se optasse por desenvolver tal serviço.

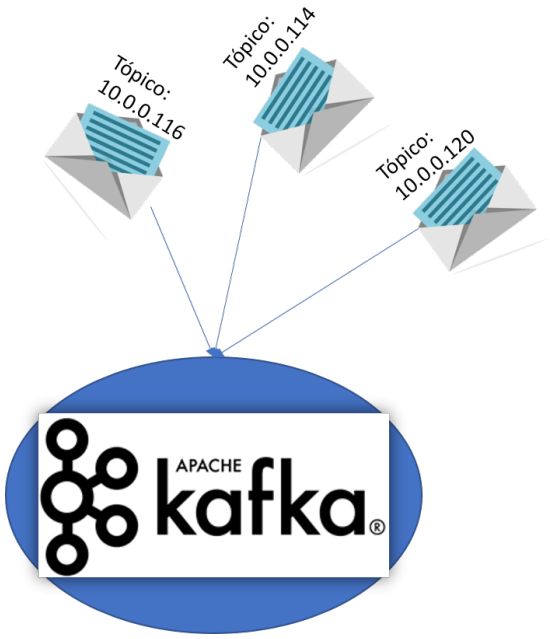
Durante as buscas por tal ferramenta que cumprisse tais exigências para servir de mecanismo de comunicação a ser usado pelo mecanismo desenvolvido durante essa pesquisa**,** encontrou-se o **Apache Kafka**, que é uma plataforma de *streaming* distribuída baseada no **Apache ZooKeeper**.

Como foi descrito na Seção 3.6, o **Apache Zookeeper** atende aos requisitos acima especificados, uma vez que possui módulos específicos em sua arquitetura para cumprir cada uma das exigências acima descritas.

Optou-se por utilizar o **Apache Zookeeper** através do **Apache Kafka** pelas funcionalidades que este último possui, conforme descrito na Seção 3.7, o que torna tal plataforma bastante atrativa para servir como mecanismo de comunicação entre os agentes.

A Figura 19 a seguir descreve como foi usado o **Apache Kafka** para atender ao objetivo de realizar a comunicação entre agentes na realização desta pesquisa.

Figura 19 – Esquema de Comunicação com o Apache Kafka



Seguindo o esquema de comunicação, utilizando o **Apache Kafka**, pode-se atender aos procedimentos descritos na Figura 18 da seguinte forma:

1) **Publicar Informações locais:** utilizando o **Apache Kafka**, o agente pode enviar o *ranking* construído localmente ao *cluster* do **Apache Kafka** com o **tópico** sendo o endereço do servidor remetente das informações.

2) **Receber Informações dos servidores de interesse:** os agentes remotos que possuem interesse em receber as informações do *ranking* construído pelo servidor que acaba de enviar uma mensagem para o **tópico**, conforme descrito no passo anterior, são notificados e atualizados utilizando a infraestrutura do **Apache Kafka** e **Apache Zookeeper**.

3) **Classificar servidores baseado nas condições de rede:** de posse das informações coletadas localmente e das informações recebidas ao verificar os **tópicos** de interesse, o agente é capaz de executar o procedimento **JointDecision** explicado na seção anterior.

4) **Tomar decisão em consenso:** ao final da execução do procedimento **JointDecision**, o agente tem sua estrutura de dados ***jointBestServer*** atualizada e, com isso, é capaz de escolher os servidores com melhores condições de rede.

5) **Armazenar a decisão tomada:** ao final, o agente armazena a decisão tomada para que ele possa utilizá-la na próxima atualização ou, caso ele seja consultado, na ocorrência de uma falha em um dos Data Centers com o qual ele se comunica.

Seguindo os procedimentos descritos ao longo deste capítulo, atingem-se todos os objetivos específicos definidos na Seção 1.2 . No próximo capítulo será descrito um Estudo de Caso no qual implementou-se o Mecanismo de Suporte à Tomada de Decisão, descrito neste capítulo, e serão mostrados alguns resultados obtidos após a execução do mecanismo desenvolvido.

5 ESTUDO DE CASO

Após definir a solução proposta para dar Suporte à Tomada de Decisão no capítulo anterior, decidiu-se empreender dois estudos de caso, utilizando a infraestrutura Local e em Nuvem, que serão descritos neste capítulo. Decidiu-se empreender um estudo utilizando uma Infreaestrutura Local a fim de demonstrar como o administrador de um conjunto de Data Centres geograficamente distribuído pode implantar a solução proposta por esta pesquisa.

Inicialmente, definiu-se uma arquitetura para implantação da solução em infraestrutura local emulando as condições das redes de alta abrangência. Em seguida, descreve-se o processo de validação realizado a fim de demonstrar a reprodutibilidade das condições de rede WANs em uma infraestrutira de rede local. Após a validação, descreve-se a arquitetura utilizada para realizar o estudo de caso em infraestrutura local. Posteriormente, descreve-se os procedimentos para implantação dos módulos da solução. Finalmente, são mostrados os resultados obtidos com os testes realizados.

5.1 Definição da infraestrutura para implantação da solução

Para reproduzir com maior fidelidade a capacidade dos links encontrados nos Data Centers, foi utilizado um roteador GigaBit ethernet, obtendo-se, assim, *throughputs* mais realistas para os cenários de uso. Para cumprir o objetivo de reproduzir condições de rede encontradas na Internet no que diz respeito aos valores de **Delay** e **Jitter**, decidiu-se utilizar o **Netem** (FOUNDATION, 2018d).

De acordo com Foundation (2018d), o **Netem** provê a funcionalidade para teste de protocolos emulando as propriedades das redes de grande abrangência. Ele é capaz de emular **delay, jitter, perda de pacotes, duplicação** e **reordenamento**.

Além de emular as condições de uma rede de alta abrangência (**WAN**), para atender ao objetivo de avaliar o **throughput** a fim de escolher os Data Centers com melhores links, precisava-se definir uma estratégia para medir a vazão da rede em tempo real. Para cumprir esse objetivo optou-se por adotar o **IPERF**.

Segundo IPERF (2017), o **Iperf3** é uma ferramenta para realizar medições ativas da largura de banda máxima atingida em redes baseadas no protocolo **IP.** Essa ferramenta é capaz de relatar em tempo real qual a vazão máxima alcançada por um determinado *link* de rede.

Após escolhermos as ferramentas necessárias para replicar as condições das redes **WAN** em redes locais (**LAN**), precisava-se montar uma infraestrutura local na qual se pudesse replicar as condições de rede. Decidiu-se, antes de coletar os resultados a serem utilizados para a tomada de decisão distribuída, montar uma infraestrutura na

qual se pudesse validar condições de rede reais previamente coletadas em uma rede de alta abrangência.

Na seção a seguir, será descrita a infraestrutura usada e os procedimentos adotados para validar a replicação de redes reais de alta abrangência em uma infraestrutura local.

5.1.1 Validação da Infraestrutura Local

Para validar a replicação de redes **WAN** em uma rede local utilizou-se a infraestrutura apresentada na Figura 20 e descrita a seguir.

Figura 20 – Infraestrutura Local

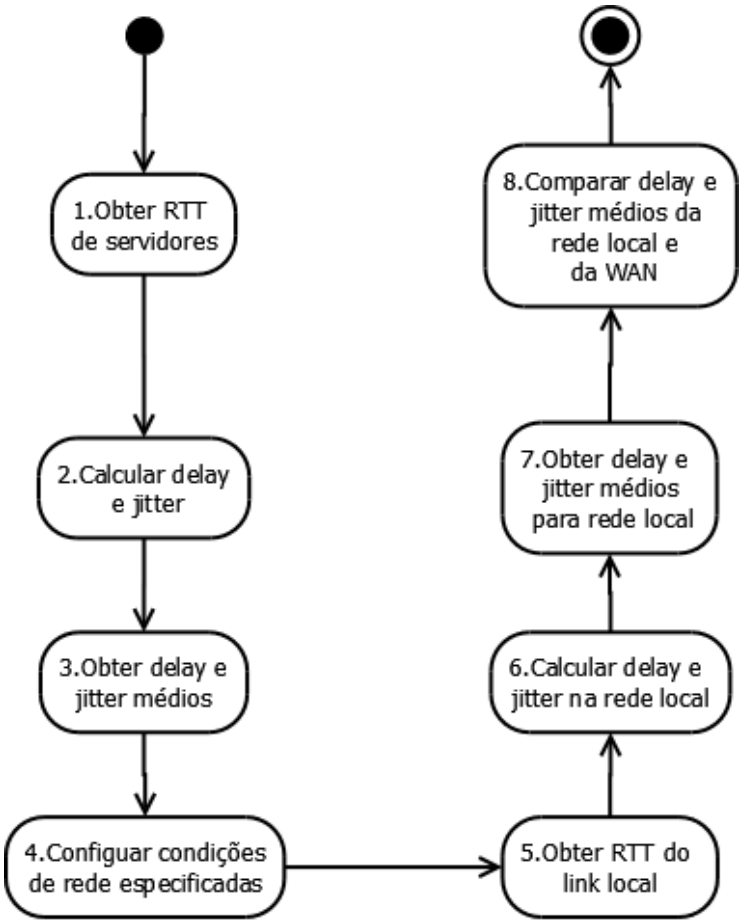
- **Cliente Iperf:** este componente é responsável por gerar tráfego TCP para a máquina que hospeda o servidor Iperf. O Iperf Client é executado em um sistema operacional Lubuntu 16.04 580 de 64 bits, em uma VM com 1Gb de RAM dentro do Xen Server 7.0.
- **Netem:** este componente é um software, Network Emulator, responsável por emular condições de rede desejadas de forma controlada. Ele permite configurar os parâmetros, Delay e Jitter, para os valores definidos em cada cenário. O software é executado na mesma máquina que o Iperf Client.
- **Roteador:** este dispositivo é responsável pelo encaminhamento dos pacotes na rede entre o cliente e o servidor. É um roteador TP-Link TL-R600VPN com quatro portas LAN Gigabit e uma porta WAN Gigabit.
- **Iperf-Server:** este componente é responsável por receber o tráfego TCP gerado pelo Iperf-Client e mensurar o throughput da comunicação. O software é executado em um sistema operacional Lubuntu 16.04 de 64 bits, em uma VM com 1Gb de RAM dentro do Xen Server 7.0.

Para realizar a emulação das condições de rede, o **Netem** permite que além de configurar os parâmetros de interesse, **Delay e Jitter**, se possa configurar uma distribuição de probabilidade à qual a variação do **Delay** possa se ajustar. Dentre as distribuições de probabilidade oferecidas pelo software de emulação (Normal, Pareto,

Pareto-normal), a mais usada para descrever a variação no **Delay**, de acordo com Foundation (2018d), é a distribuição normal. Optou-se, então, por utilizar essa distribuição de probabilidade com parâmetros: $\mu = Delay$ e $\sigma = Jitter$.

Para validar a infraestrutura local, adotou-se o procedimento mostrado no fluxograma da Figura 21 descrito a seguir.

Figura 21 – Procedimento de validação das condições WAN em rede LAN



1) **Obter RTT dos servidores:** neste passo, utilizou-se o servidor na nuvem Cloud At Cost para disparar requisições de ping para cada um dos servidores usados na realização do planejamento de experimento, a fim de obter o Round Trip Time entre o servidor utilizado e cada um dos sites.

2) **Calcular delay e jitter:** neste passo, utilizaram-se as relações mostradas na Equações 3.1 e 3.2 com cada um dos valores de RTT obtidos no passo anterior, a fim de obter as amostras de delay e jitter a partir do RTT.

3) **Obter delay e jitter médios:** a partir dos valores de delay e jitter obtidos no passo anterior, aplicaram-se as relações mostradas na Equação 4.1, a fim de obter os valores médios para o delay e para o jitter.

4) **Configurar condições de rede especificadas:** neste passo, a partir dos valores médios de delay e jitter obtidos anteriormente, configurou-se a ferramenta **Netem** para emular a condição de rede com um determinado delay e jitter médios.

5) **Obter o RTT do link local**: repetiu-se o procedimento adotado no passo 1 utilizando-se um computador conectado à rede local a fim de obter os valores de RTT com uma dada condição de rede previamente configurada.

6) **Calcular o delay e o jitter na rede local:** com os valores de RTT obtidos no passo anterior, repetiu-se o procedimento adotado no passo 2, a fim de obter os valores de delay e jitter para a medição local.

7) **Obter delay e jitter médios para a rede local:** neste passo, realizou-se o mesmo procedimento aplicado no passo 3 para obter os valores de delay e jitter médios para as amostras de RTT obtidas na infraestrutura local.

8) **Comparar delay e jitter médios da rede local e da WAN:** após realizados todos os procedimentos anteriores, foram obtidas duas amostras de Delay e Jitter, uma na WAN e outra na rede local modificada pelo **Netem**. Utilizou-se o procedimento de *Bootstrapping* para calcular os intervalos de confiança para as médias obtidas e em seguida realizou-se o teste não paramétrico de Kolmogorov Smirnov, a fim de comparar e verificar se os valores médios de delay e jitter obtidos na Internet são estatisticamente iguais aos obtidos em rede local.

Ao realizar testes utilizando o **Netem** para um determinado cenário de coleta na Internet, obtém-se o resultado mostrado na Tabela 5:

Tabela 5 – Medianas do RTT- Experimento na WAN X Rede Local Com Netem

Cenário	Mediana RTT WAN(ms)	Mediana RTT Netem(ms)	Média RTT WAN(ms)	Média RTT Netem(ms)
Google	7.9	8.75	7.80	8.71
Amazon CA	27	28.2	27.03	28.2
UOL	157.8	160	158.17	159.55

Como se pode perceber pelos resultados mostrados na Tabela 5**,** há uma aproximação bastante razoável entre as condições testadas na rede de grande abrangência (**WAN**) e as condições de rede testadas em rede local alterada pelo software **Netem**.

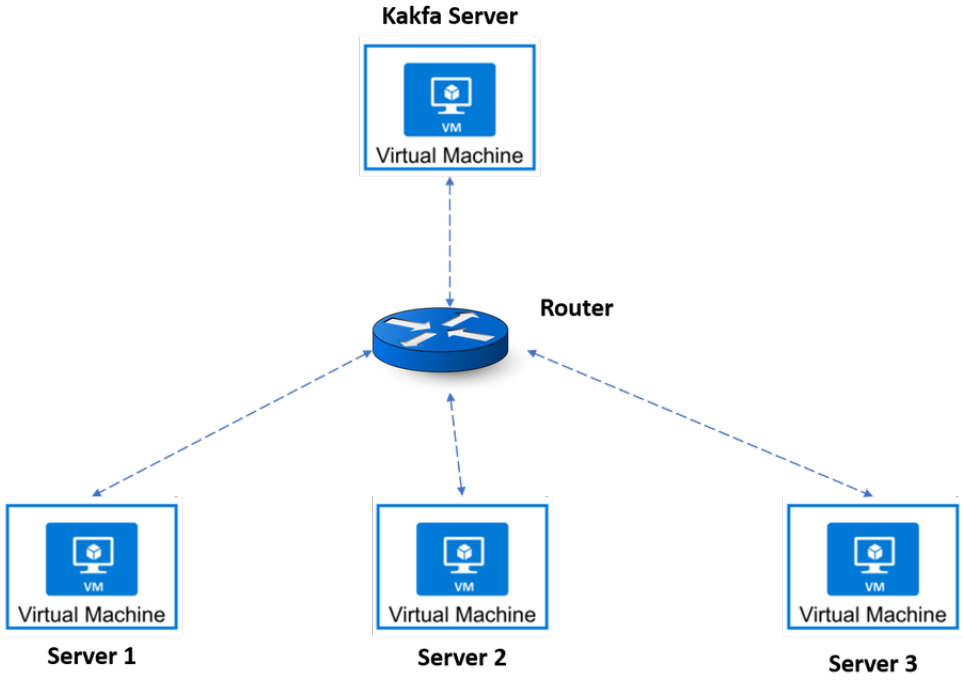
Ao validar-se a reprodução das condições de rede de alta abrangência (**WAN**) em uma rede local, podem-se começar a implantar os módulos que irão compor a arquitetura final da solução desenvolvida durante esta pesquisa.

A linguagem escolhida para implementar os módulos foi Python, por ser nativamente suportada em ambientes Linux que são predominantes nas infraestruturas dos Data Centers. Na próxima seção, serão descritos os procedimentos adotados para construir uma infraestrutura onde se pudesse testar os módulos da solução desenvolvida durante esta pesquisa.

5.1.2 Infraestrutura para realização do estudo de caso em uma infraestrutura de rede local

Fez-se necessário montar uma infraestrutura sobre a qual os algoritmos responsáveis por executar os procedimentos de cada módulo a ser desenvolvido pudessem executar. A Figura 22 representa a construção lógica da infraestrutura usada para testar a solução desenvolvida.

Figura 22 – Infraestrutura usada para testar a solução final



Como mostrado na Figura 22, o ambiente montado para validar a solução desenvolvida durante esta pesquisa é composto por 3 VMs (**Server 1, Server 2, Server3**) que emulam 3 Data Centers geograficamente distribuídos, um roteador Gigabit Ethernet TP-Link TL-R600VPN e 1 VM na qual é implantado o **Kafka Server**.

As condições dos links foram estipuladas a partir de testes realizados na Internet com medições a partir de uma máquina virtual implantada no serviço **CloudAtCost** para três sites distintos, a saber:

- Google
- Amazon CA e
- UOL

Esses cenários foram os escolhidos por apresentarem condições de rede bastante distintas. As condições de rede desses links são mostradas na Tabela 6 .

Tabela 6 – Cenários de Teste da Solução

	Delay	**Jitter**
Google	3.85	0.63
Amazon CA	13.55	0.72
UOL	79.51	1.83

O *cluster* **Apache Kafka** utilizado para testar a solução possui comunicação com todas as VMs representativas dos Data Centers. A comunicação entre as VMs que representam os Data Centers é a seguinte:

- **Server 1 ->> Server2**
- **Server 1 ->> Server3**
- **Server 2->> Server 1**
- **Server 2 ->> Server 3**
- **Server 3->> Server 1**
- **Server 3->> Server 2**

Com isso, podemos integrar os módulos da solução e cumprir os objetivos estabelecidos na Seção 1.2. Na seção a seguir, será descrito o procedimento adotado para a implementação do módulo de medição de rede.

5.1.3 Implantação do Módulo de Análise de rede

A primeira decisão a ser tomada para a implantação do módulo de análise de rede, conforme descrito na Seção 4.2.4, seria a definição da ferramenta a ser usada para obter o RTT. Optou-se por utilizar a ferramenta **Ping** por ser mais flexível quanto à utilização de qualquer sistema operacional.

Outra decisão a ser feita era a definição dos servidores que iriam se comunicar e, para atender a essa necessidade, precisava-se encontrar uma forma de definir os endereços para que as condições de rede para tal servidor pudessem ser testadas. Diante disso, optou-se por utilizar-se uma string contendo os endereços dos servidores comunicantes separados por virgula.

Para atender ao objetivo de definir a quantidade de amostras, construiu-se um procedimento que fosse capaz de receber esse valor através de um parâmetro, tornando assim o mecanismo de medição mais flexível.

Após a construção de um mecanismo de análise totalmente parametrizado, decidiu-se empreender a construção de um arquivo de configuração com o qual o administrador de sistemas pudesse futuramente trocar essas definições sem a necessidade de reimplementar o mecanismo de análise de rede. Dessa forma, definiu-se a estrutura mostrada no código 5.1 abaixo e descrito a seguir.

Código 5.1 – Exemplo de arquivo de configuração do mecanismo de medição

```
<config>
   <rttTool id="ping"/>
   <time updateTime="10"/>
   <servers>
     <serversList numberOfServers="3" addresses="
        10.0.0.115,10.0.0.116,10.0.0.117" numberOfSamples="5"
        />
   </servers>
</config>
```

- **config**: *tag* utilizada para marcar o início de uma configuração.
- **rttTool:** essa marcação é utilizada para definir a ferramenta a ser utilizada para realizar a coleta do RTT nos links entre o *Data Center* no qual o mecanismo está em execução e o *Data Center* remoto.
- **time:** Essa marcação é utilizada para informar o intervalo de tempo em segundos a ser utilizado pelo agente para realizar uma medição das condições de rede local e para verificar se há informações a serem coletadas do **Apache Kafka**.
- **servers:** *tag* utilizada para marcar o início da configuração da comunicação com servidores remotos.
- **serversList:** marcação utilizada para especificar a lista de servidores com os quais o mecanismo irá testar o *link* e as definições a serem utilizadas na medição.
  - **numberOfServers**: propriedade utilizada para especificar o número de *Data Centers* com os quais o *Data Center* no qual o mecanismo está implantando mantém contato.
  - **addresses:** propriedade utilizada para definir os endereços dos *Data Centers* remotos com os quais o *Data Center* no qual o mecanismo está implantado mantém contato.

- **numberOfSamples**: propriedade utilizada para especificar o número de amostras de RTT a serem coletadas para análise.

Com o arquivo de configuração definido no código 5.1, atende-se ao objetivo de definir um mecanismo de análise de rede flexível, conforme descrito na Seção 4.2.2 .

Após a definição do mecanismo de análise de rede precisava-se implantar o módulo capaz de utilizar o Apache Kafka para realizar a comunicação entre os agentes, conforme será descrito na próxima seção.

5.1.4 Implantação do módulo de comunicação entre agentes

Para cumprir o objetivo de definir uma estratégia de comunicação utilizando o **Apache Kafka**, precisava-se atender aos seguintes requisitos:

1) Definir uma estratégia de comunicação transparente com o **Apache Kafka**
2) Definir uma estratégia de comunicação *multithreading* que permita ter um canal de comunicação síncrono capaz de receber mensagens dos agentes com os quais um determinado *Data Center* se comunica e
3) Definir uma estratégia de comunicação que seja capaz de possibilitar o envio de mensagens informando as condições de rede locais para os outros agentes com os quais um determinado *Data Center* se comunica.

Para definir uma estratégia transparente de comunicação com o **Apache Kafka**, decidiu-se utilizar a biblioteca Python **PyKafka**. De acordo com Parsely (2018), o **PyKafka** é um cliente para o **Apache Kafka** em Python e inclui implementações para produtores e consumidores, que possuem opcionalmente um *back-end* implementado em linguagem C construída sobre o **librdkafka**.

Para atender ao requisito de estabelecer uma estratégia de comunicação *multithreading* que permita ter um canal de comunicação síncrono capaz de receber mensagens, resolveu-se utilizar a implementação **Thread** da biblioteca **threading** do Python. Segundo Python (2018), a classe **Thread** representa uma atividade que executa numa *thread* de controle separada.

Para concluir o mecanismo de comunicação, precisava-se estabelecer uma estratégia de comunicação que possibilitasse aos agentes enviar mensagens para outros agentes com os quais eles se comuniquem de tal forma que fosse possível aos agentes enviar as condições de rede locais e receber as informações de rede remotas dos outros agentes.

Para cumprir esse objetivo, precisava-se cumprir algumas exigências, a saber:

1) Definir uma identificação única para cada agente a fim de possibilitar a ele enviar mensagens ao **Apache Kafka.**

2) Estabelecer uma forma de utilizar a estrutura de **tópicos** proposto pelo **Apache Kakfa** para que os agentes pudessem enviar mensagens ou se inscrever para receber as mensagens apropriadamente.

Para cumprir a primeira exigência, resolveu-se utilizar o endereço IP do *Data Center* no qual o agente está implantado de modo que ele possa ser identificado unicamente.

A segunda exigência é cumprida utilizando a identificação única do agente para gerar tópicos de interesse. Desse modo, ao enviar uma mensagem para o **Apache Kafka**, o agente utiliza a identificação única como tópico de modo que os agentes com os quais ele se comunica possam se inscrever para receber as mensagens enviadas por ele utilizando o IP dele.

Ao final, tinha-se um mecanismo de comunicação entre agentes implementados em Python utilizando o **PyKafka** e a biblioteca padrão **Threading**. Ao inicializar o módulo de comunicação, o agente informa os endereços IPs dos *Data Centers* com os quais ele mantém contato de modo que ele se inscreve para receber mensagens nos **tópicos** relacionados aos IPs e cria um tópico com o IP do *Data Center* no qual ele está implantado de modo que outros agentes possam se inscrever para receber mensagens dele.

A fim de tornar configurável a identificação do agente e os dados de acesso aos servidores do **Apache Kafka**, resolveu-se incrementar o arquivo de configuração de modo a incluir esses parâmetros. O código 5.2 mostra o arquivo de configuração atualizado.

Código 5.2 – Exemplo do arquivo de configuração do agente atualizado.

```
<config>
   <agent id="10.0.0.115"/>
   <rttTool id="ping"/>
   <time updateTime="10"/>
   <servers>
     <kafka address="10.0.0.118" port="9092"/>
     <serversList numberOfServers="2" addresses="
        10.0.0.116,10.0.0.117" numberOfSamples="5"/>
   </servers>
</config>
```

No código mostrado acima adicionou-se uma nova *tag,* ***kafka****,* com dois parâmetros: **address** e **port**. O parâmetro **address** especifica o endereço do servidor do **Apache Kafka** e o parâmetro **port** especifica a porta utilizada no servidor remoto para

comunicar-se com o **Apache Kafka**. Além disso, tem-se a *tag* **agent** que possui a identificação do agente que será usada como tópico no **Apache Kafka** para envio de mensagens.

Com isso tem-se um arquivo de configuração com todos os parâmetros a serem utilizados pelos módulos de Análise de Rede e de Comunicação Distribuída, tendo desse modo todas as ferramentas necessárias para a construção do Agente que será explicada na próxima seção.

5.1.5 Implantação do Agente

Após a implantação do módulo de análise de rede e de comunicação distribuída entre agentes, empreendeu-se a implantação do algoritmo final do agente de modo a cumprir o objetivo da arquitetura mostrada na Figura 8. O agente deve, em conjunto com os outros dois módulos previamente construídos, corroborar para prover o suporte à tomada de decisão.

Desse modo, para fins de testar essa arquitetura, decidiu-se construir um algoritmo em Python que executasse *em um Shell* do Linux realizando medições locais e recebendo medições realizadas por outros agentes de modo a construir um *ranking* geral.

Para construir o *ranking* geral de forma que a decisão proferida por cada um dos agentes possuísse igual relevância na tomada de decisão global, resolveu-se, após a construção do *ranking* local considerando a vazão dos links, enviar para os outros agentes apenas a colocação dos *Data Centers* sem informar as vazões.

Ao receber os *rankings* feitos pelos agentes remotos, cada agente monta o *ranking* global ponderando a quantidade de vezes que o *Data Center* apareceu com sua colocação e dividindo o resultado final pelo número de servidores. A ponderação usada é mostrada na Equação 5.1.

$$rankingGeral['idAgente'] = \sum_{i=0}^{N} \frac{\frac{1}{i+1}}{NServers} \tag{5.1}$$

Desse modo, o *Data Center* recebe uma pontuação que depende de sua colocação **"i"** e do número de servidores "**NServers**".

Cada um dos agentes usa apenas as informações recebidas de outros agentes para montar o *ranking* geral, pois, dessa forma, elimina-se a possibilidade de uma decisão viesada por considerar as suas análises locais duas vezes (uma direta e outra indireta).

Com isso, termina-se a implantação do mecanismo proposto por esta pesquisa.

Portanto, atinge-se o objetivo estabelecido no início deste trabalho: auxiliar o administrador de sistema na tomada de decisão para a migração de VMs de forma a minimizar o tempo de restauração de serviços de um *Data Center* caso ele venha falhar.

Na próxima seção, serão mostrados alguns resultados obtidos com a execução do agente desenvolvido.

5.2 Resultados

A fim de mostrar alguns resultados da execução dos agentes distribuídos em nossa infraestrutura, decidimos criar um cliente baseado em Linux *Bash Shell* para que pudéssemos, principalmente, verificar o *ranking* geral construído por três agentes que executaram em condições de rede diversas. Os resultados obtidos são mostrados nas figuras a seguir.

Figura 23 – Tela do Data Center emulado 1

```
--------ID:192.168.0.102-------------
[('192.168.0.102', 2.5), ('192.168.0.104', 0.75), ('192.168.0.103', 0.5)]
--------ID:192.168.0.102-------------
[('192.168.0.102', 5.5), ('192.168.0.104', 1.5), ('192.168.0.103', 1.25)]
--------ID:192.168.0.102-------------
[('192.168.0.102', 9.0), ('192.168.0.104', 2.5), ('192.168.0.103', 2.0)]
--------ID:192.168.0.102-------------
[('192.168.0.102', 13.5), ('192.168.0.104', 3.75), ('192.168.0.103', 3.0)]
--------ID:192.168.0.102-------------
[('192.168.0.102', 18.5), ('192.168.0.104', 5.0), ('192.168.0.103', 4.25)]
--------ID:192.168.0.102-------------
[('192.168.0.102', 25.5), ('192.168.0.104', 6.75), ('192.168.0.103', 6.0)]
```

Figura 24 – Tela do Data Center emulado 2

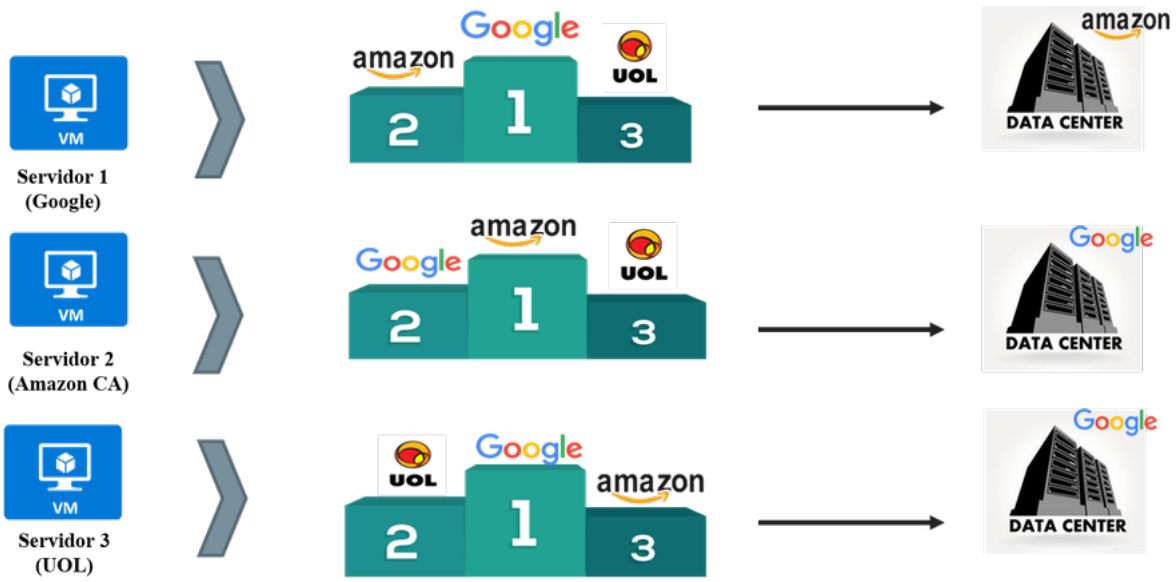
```
--------ID:192.168.0.104-------------
[('192.168.0.104', 1.5), ('192.168.0.102', 1.0), ('192.168.0.103', 0.5)]
--------ID:192.168.0.104-------------
[('192.168.0.104', 3.25), ('192.168.0.102', 2.5), ('192.168.0.103', 1.0)]
--------ID:192.168.0.104-------------
[('192.168.0.104', 6.25), ('192.168.0.102', 4.5), ('192.168.0.103', 2.0)]
--------ID:192.168.0.104-------------
[('192.168.0.104', 9.5), ('192.168.0.102', 7.0), ('192.168.0.103', 3.0)]
--------ID:192.168.0.104-------------
[('192.168.0.104', 13.75), ('192.168.0.102', 9.5), ('192.168.0.103', 4.5)]
--------ID:192.168.0.104-------------
[('192.168.0.104', 18.25), ('192.168.0.102', 12.5), ('192.168.0.103', 6.0)]
```

Figura 25 – Tela do Data Center emulado 3

```
--------ID:192.168.0.103------------
[('192.168.0.102', 0.5), ('192.168.0.103', 0.25)]
--------ID:192.168.0.103------------
[('192.168.0.102', 1.0), ('192.168.0.103', 0.75), ('192.168.0.104', 0.5)]
--------ID:192.168.0.103------------
[('192.168.0.102', 2.0), ('192.168.0.103', 1.75), ('192.168.0.104', 1.5)]
--------ID:192.168.0.103------------
[('192.168.0.102', 3.5), ('192.168.0.103', 3.25), ('192.168.0.104', 3.0)]
--------ID:192.168.0.103------------
[('192.168.0.102', 5.5), ('192.168.0.103', 5.25), ('192.168.0.104', 5.0)]
--------ID:192.168.0.103------------
[('192.168.0.102', 8.5), ('192.168.0.103', 8.25), ('192.168.0.104', 8.0)]
--------ID:192.168.0.103------------
[('192.168.0.102', 12.0), ('192.168.0.103', 11.5), ('192.168.0.104', 11.0)]
```

Baseado nas figuras mostradas acima, é possível perceber que dois dos três agentes (DC 1 e DC 3), cada um implantado em uma máquina virtual que emula um *Data Center*, chegam a um consenso e conseguem mostrar o mesmo servidor na primeira colocação do *ranking*. Essa diferença em relação ao DC 2 se deve às condições de rede locais emuladas que pioram o *Delay* local, prejudicando a comunicação, o que acontece de forma semelhante na WAN. Esses resultados mostram que é possível escolher o melhor DC, dadas as condições de rede locais, conforme apresentado na Figura 26.

Figura 26 – Ranking



Além dos resultados obtidos na infraestrutura local, também executamos o mecanismo na Internet para verificarmos o comportamento do mesmo. Assim, conseguiu-se comprovar que os agentes foram capazes de trocar informações sobre condições de rede em um cenário distribuído, de modo que a solução pode ser utilizada em situações

reais para auxiliar no processo de tomada de decisão quanto à escolha do DC com melhor vazão. A Tabela 7 apresenta os resultados do *raking* global de um agente localizado em um DC nos Estados Unidos, no qual pode-se verificar que foi possível ordenar as vazões de forma decrescente indicando que, em caso de falha desse DC, o site que possui maior vazão para atendê-lo é o da Google.

Tabela 7 – Resultado da Solução na WAN

Site	**RTT (ms)**	**Throughput(Mbps)**
Google	6.29	2900.67
Adidas ZA	402.60	1340.40
OI	382.74	251.53
Harvard	29.69	129.13
Amazon CA	26.27	98.93
Oxford	94.77	40.10
Amazon UK	93.92	28.74
Sorbonne	126.63	26.29
Loot ZA	229.93	23.54
Uol	362.76	12.97

Optamos por enfatizar a construção do mecanismo e mostrar apenas alguns resultados, devido ao fato de não encontrarmos uma solução semelhante a nossa de modo que pudéssemos comparar os resultados e informar ao leitor a solução que se sobressai.

Dessa forma, conseguiu-se demonstrar o funcionamento da estratégia desenvolvida durante a realização dessa pesquisa, comprovando que os objetivos estabelecidos na Seção 1.2 foram cumpridos.

6 CONSIDERAÇÕES FINAIS

Com a popularização do paradigma de computação em nuvem e a crescente migração de serviços críticos para esse ambiente, a disponibilidade tem se tornado um de seus principais desafios (NABI; TOEROE; KHENDEK, 2016). Nesse contexto, em que os serviços de *cloud computing* são tão relevantes para o cenário socioeconômico, é de suma importância garantir a disponibilidade e a conformidade com os SLAs, através de ações pró-ativas que minimizem o tempo para reparar o serviço fornecido por um data center em caso de falhas.

Os data centers são grandes edifícios que hospedam diversos componentes, tais como hardware, software, infraestrutura de rede, armazenamento. Todos esses componentes podem falhar e afetar o desempenho de aplicações ou até mesmo interromper a execução do sistema (CHEN; GAO; CHEN, 2016) (ZAINELABDEN et al., 2016).

Nessa perspectiva, esta pesquisa propôs um mecanismo multiagente para auxiliar na escolha entre um conjunto de centros de dados geograficamente distribuídos escolhendo aquele que ofereça a melhor vazão de links para restabelecer os serviços de um data center com defeito, conforme discutido no Capítulo 1. Para atingir os objetivos, foi definido um procedimento metodológico para o desenvolvimento da solução proposta.

O principal objetivo dessa solução é monitorar os links que interligam um conjunto de data centers geograficamente distribuídos, através de agentes que se comunicam para trocar informações sobre a vazão dos links, e fornecer aos administradores de data center informações precisas que os auxiliem na tomada de decisão para recuperação de serviços (redirecionamento de tráfego) ou para a realização de backups distribuídos.

O mecanismo é composto por três módulos: 1. Módulo de Análise de rede; 2. Agente para tomada de decisão e 3. Módulo de Comunicação. O primeiro módulo é responsável por mensurar condições de rede (obter o RTT, calcular o delay, o jitter e a vazão); no módulo 2, os agentes utilizam o mecanismo de análise de rede para mensurar a sua vazão e a dos demais data centers com os quais ele possui ligação (links), e criar um *ranking* com os data centers que possuam melhor vazão para lhe atender; por último, no módulo 3, os agentes precisam compartilhar suas informações entre si, através da plataforma de mensagem Kafka, e, com base na decisão de cada agente, é possível chegar a um consenso por meio do método de votação por maioria simples.

Para desenvolver o mecanismo de análise de rede, precisava-se, antes, validar

a infraestrutura local montada na tentativa de emular as condições de rede de alta abrangência. Para tal, utilizou-se o software **Netem** de modo que se obtivesse valores de delay e jitter semelhantes aos encontrados na WAN. Após montagem e validação dessa infraestrutura local, constatou-se que foi atingido o objetivo estabelecido de replicar condições de rede (bem próximas da Internet) em infraestrutura local, tornando possível a construção de agentes capazes de *rankear Data Centers* baseado em condições de rede.

Desse modo, desenvolveu-se duas abordagens de medição de rede, local e em nuvem, de modo a possibilitar a escolha por parte do administrador do *Data Center* do módulo que melhor atenda às suas necessidades. O módulo de comunicação de agentes distribuídos foi implementando através da utilização do **Apache Kafka**, uma vez que ele possibilita que haja comunicação entre servidores de interesse através do mecanismo **Publish and Subscribe**.

Os resultados obtidos através da aplicação do mecanismo no estudo de caso, comprovam que a solução pode ser utilizada em situações reais e auxiliar no processo de tomada de decisão dos administradores de rede, no que tange à escolha do site que possua melhor vazão.

Desta forma, foi possível alcançar os objetivos estabelecidos no Capítulo 1, visto que se definiu uma metodologia de medição e análise da rede, que forneceu subsídios para o desenvolvimento dos agentes, e a implantação de uma arquitetura distribuída para o compartilhamento de mensagens da solução multiagente.

E como contribuição principal temos o mecanismo como produto final desta pesquisa, o qual é capaz de monitorar e analisar a rede de um *pool* de data centers distribuídos em tempo real, rankear a vazão de cada site com apoio de agentes e compartilhar a decisão tomada por cada agente para que se tenha um consenso para uma decisão mais assertiva. Essa decisão diz respeito ao data center que apresente melhores condições de rede (vazão) para atender a um determinado site em condição de falha.

Quanto às limitações desta pesquisa, é importante destacar que esse mecanismo é um ponto de apoio para a decisão final dos administradores de data centers na migração de dados ou redirecionamento de tráfego. Ele não deve ser entendido como uma solução suficiente para migrar ou redirecionar tráfego em caso de falha de determinado site. Além disso, este trabalho não se propõe a fornecer um mecanismo de migração de VMs (*Live migration*), ele apenas fornece informações sobre a vazão dos links para que depois seja executada a migração.

Embora este trabalho tenha alcançado alguns resultados, novos estudos ainda podem ser empreendidos a partir dele, tais como:

- Agregar uma estratégia de *Live migration* ao mecanismo, de modo que os agentes sejam capazes de apontar o data center com melhor vazão e executar a migração.
- Fazer uma análise de capacidade dos centros de dados para verificar o estado de recursos como processador, memória e armazenamento disponível.
- Construir modelos de performabilidade que visem minimizar o MTTR de serviços de data centers.

Espera-se, portanto, que os trabalhos futuros aprimorem a abordagem inicial do mecanismo, agregando novas ideias para tornar a solução mais robusta. Desse modo, pretende-se contribuir com a garantia da disponibilidade dos serviços de nuvem, uma vez que tais serviços estão cada vez mais se tornando tão críticos para a sociedade atual.

REFERÊNCIAS

AMAZON. *Summary of the Amazon S3 Service Disruption in the Northern Virginia (US-EAST-1) Region*. 2017. Disponível em: <https://aws.amazon.com/pt/message/41926/>. Acesso em: 20/09/2018.

ANDRADE, E. C. de. *Modelagem e Análise de Mecanismos de Tratamento de Interrupções em Infraestruturas Computacionais dos Sistemas Distribuídos*. 2014. 157 p. Tese (Pós-Graduação em Ciência da Computação) — Universidade Federal de Pernambuco, Recife. Disponível em: <http://www.modcs.org/wp-content/uploads/thesis/Tese-Ermeson.pdf>. Acesso em: 10/02/2018.

ARIF, M.; KIANI, A. K.; QADIR, J. Machine learning based optimized live virtual machine migration over WAN links. *Telecommunication Systems*, Springer US, 2016. ISSN 15729451.

ARMBRUST, M. et al. *Above the Clouds*: A Berkeley View of Cloud Computing. [S.l.], 2009. Disponível em: <http://bit.ly/2qvwFAW>.

AWS, A. *What is Pub/Sub Messaging?* 2018. Disponível em: <https://aws.amazon.com/pt/pub-sub-messaging/>. Acesso em: 25/05/2018.

BAUER, E.; ADAMS, R.; EUSTACE, D. *Beyond redundancy*: how geographic redundancy can improve service availability and reliability of computer-based systems. [S.l.]: John Wiley & Sons, 2011.

BEYER, B. et al. *Engenharia de Confiabilidade do Google*: Como o Google administra seus sistemas de produção. [S.l.]: Novatec Editora, 2016.

BISWAS, M. I. et al. A practical evaluation in openstack live migration of VMs using 10Gb/s interfaces. In: IEEE, 2016. *Service-Oriented System Engineering (SOSE), 2016 IEEE Symposium on*. [S.l.], 2016. p. 346 – 351.

CHEN, T.; GAO, X.; CHEN, G. The features, hardware, and architectures of data center networks: A survey. *Journal of Parallel and Distributed Computing*, Elsevier Inc., v. 96, p. 45 – 74, 2016. ISSN 07437315. Disponível em: <http://dx.doi.org/10.1016/j.jpdc.2016.05.009>.

COULOURIS, G. et al. *Sistemas Distribuídos*: Conceitos e Projeto. 5ª. ed. [S.l.]: Bookman, 2013.

DIALLO, M. H. et al. 4 - AutoMigrate: A Framework for Developing Intelligent, SelfManaging Cloud Services with Maximum Availability. In: *2016 International Conference on Cloud and Autonomic Computing (ICCAC)*. [s.n.], 2016. p. 95 – 106. ISBN 978-1-5090-3536-6. Disponível em: <http://ieeexplore.ieee.org/document/7774964/>.

DUGGAN, M.; DUGGAN, J.; BARRETT, E. 3 - An Autonomous Network Aware VM Migration Strategy in Cloud Data Centres. *2016 International Conference on Cloud and Autonomic Computing (ICCAC)*, IEEE, p. 24 – 32, sep 2016. Disponível em: <http://ieeexplore.ieee.org/document/7774957/>.

FOROUZAN, B. A. *Comunicação de Dados e Redes de Computadores*. [S.l.: s.n.], 2007. ISBN 9788563308474.

FOUNDATION, A. *Apache Kafka: A distributed streamming platform*. 2018. Disponível em: <http://bit.ly/2HPV8Hz>. Acesso em: 15/06/2018.

FOUNDATION, A. *ZooKeeper: A Distributed Coordination Service for Distributed Applications*. 2018. Disponível em: <http://bit.ly/2JMQpIE>. Acesso em: 15/06/2018.

FOUNDATION, L. *hping3 - send (almost) arbitrary TCP/IP packets to network hosts*. 2018. Disponível em: <http://bit.ly/2JnxELK>. Acesso em: 07/08/2018.

FOUNDATION, L. *NetEm - Network Emulator*. 2018. Accessed in: Apr./2018. Disponível em: <http://bit.ly/2Hmpghx>.

FOUNDATION, L. *ping, ping6 - send ICMP ECHO_REQUEST to network hosts*. 2018. Disponível em: <http://bit.ly/2JuaGCr>. Acesso em: 07/06/2018.

FOUNDATION, L. *Token Bucket Filter*. 2018. Accessed in: Apr./2018. Disponível em: <http://bit.ly/2H0yIUo>.

GARTNER. *Gartner Says Worldwide IaaS Public Cloud Services Market Grew 31 Percent in 2016*. STAMFORD: [s.n.], 2017. Disponível em: <https://www.gartner.com/newsroom/id/3808563>. Acesso em: 30/05/2018.

GIL, A. C. *Como Elaborar projetos de Pesquisa*. 4. ed. [S.l.]: Atlas, 2002.

GOODRICH, M.; TAMASSIA, R. *Estruturas de Dados & Algoritmos em Java - 5ed*. [S.l.]: Bookman Editora, 2013. ISBN 9788582600191.

GROUP, I. N. W. *Benchmarking Terminology for Network Interconnection Devices*. 1991. Disponível em: <https://www.ietf.org/rfc/rfc1242.txt>. Acesso em: 19/10/2018.

IDC. *Worldwide Public Cloud Services Spending Forecast to Reach $122.5 Billion in 2017, According to IDC*. FRAMINGHAM: [s.n.], 2017. International Data Corporation. Disponível em: <https://www.idc.com/getdoc.jsp?containerId=prUS42321417>. Acesso em: 30/05/2018.

IPERF. *iPerf - The ultimate speed test tool for TCP, UDP and SCTP*. 2017. Disponível em: <https://iperf.fr/>. Acesso em: 10/08/2017.

ISO/IEC-27002. *Information technology — Security techniques — Code of practice for information security management*. [S.l.], 2005. Disponível em: <http://bcc.portal.gov.bd/sites/default/files/files/bcc.portal.gov.bd/page/adeaf3e5_cc55_4222_8767_f26bcaec3f70/ISO_IEC_27002.pdf>. Acesso em: 30/05/2018.

JAIN, R. *The Art of Computer Systems Performance Analysis*: Techniques for Experimental Design, Measurement, Simulation, and Modeling. [S.l.]: Wiley, 1990. ISBN 9780471503361.

JAMMAL, M. et al. Mitigating the Risk of Cloud Services Downtime Using Live Migration and High Availability-Aware Placement. In: IEEE, 2016. *Cloud Computing Technology and Science (CloudCom), 2016 IEEE International Conference on*. [S.l.], 2016. p. 578 – 583.

JURISTO, N.; M. MORENO, A. *Basics of Software Engineering Experimentation*. [S.l.]: Springer Science & Business Media, 2013.

KAPIL, D.; PILLI, E. S.; JOSHI, R. C. Live virtual machine migration techniques: Survey and research challenges. *Proceedings of the 2013 3rd IEEE International Advance Computing Conference, IACC 2013*, p. 963 – 969, 2013.

KUROSE, J. F.; ROSS, K. W. *Redes de Computadores e a Internet*: Uma abodagem Top-Down. 5ª. ed. [S.l.]: PEARSON, 2010. ISBN 978-85-88639-97-3.

LEE, D. *Amazon data centre fault knocks websites offline temporarily*. 2017. Disponível em: <http://www.bbc.com/news/world-us-canada-39119089>. Acesso em: 31/05/2018.

LU, X. et al. JTangCMS: An efficient monitoring system for cloud platforms. *Information Sciences*, Elsevier Inc., v. 370-371, p. 402 – 423, 2016. ISSN 00200255. Disponível em: <http://dx.doi.org/10.1016/j.ins.2016.06.009>.

MACIEL, P. R. M. Modeling Availability Impact in Cloud Computing. In:_____. *Principles of Performance and Reliability Modeling and Evaluation*. [S.l.]: Springer International Publishing Switzerland, 2016.

MATHEWS, P. G. *Design of Experiments with MINITAB*. [S.l.]: ASQ Quality Press, 2005.

MINITAB. *Interpretar os principais resultados para Gráfico de interação*. 2018. Disponível em: <http://bit.ly/2sxqNsM>. Acesso em: 05/06/2018.

MINITAB. *O que é um gráfico de efeitos principais?* 2018. Disponível em: <http://bit.ly/2Lr96lD>. Acesso em: 05/06/2018.

NABI, M.; TOEROE, M.; KHENDEK, F. Availability in the cloud: State of the art. *Journal of Network and Computer Applications*, v. 60, p. 54 – 67, 2016. ISSN 10958592.

NORVIG, P. ; RUSSELL, S. Inteligência Artificial: Tradução da 3 Edição. 3 ed. Rio de Janeiro: 2014.

PARSELY, I. *PyKafka — pykafka 2.8.0-dev.3 documentation*. 2018. Disponível em: <http://bit.ly/2K5SOBB>. Acesso em: 19/06/2018.

PERSICO, V. et al. On the performance of the wide-area networks interconnecting public-cloud datacenters around the globe. *Computer Networks*, Elsevier, v. 112, p. 67 – 83, 2017.

PYTHON. *threading — Thread-based parallelism — Python 3.6.6rc1 documentation*. 2018. Disponível em: <http://bit.ly/2ylYBhR>. Acesso em: 19/06/2018.

RUSSELL, S.; NORVIG, P.; DAVIS, E. *Artificial Intelligence*: A Modern Approach. [S.l.]: Pearson Education, Limited, 2016. (Always learning). ISBN 9781292153964.

SANTOS JUNIOR, A. L. D. *Integração de Sistemas com Java*. [S.l.]: Brasport, 2007.

TEYEB, H. et al. Traffic-aware Virtual Machine Migration Scheduling Problem in Geographically Distributed Data Centers. *IEEE 9th International Conference on Cloud Computing Traffic-aware*, 2016.

TONCAR, V. *VoIP Basics*: About Jitter. 2018. Accessed in: Apr./2018. Disponível em: <https://goo.gl/3v85Py>.

WAZLAWICK, R. *Metodologia de pesquisa para ciência da computação*. 2. ed. [S.l.]: Elsevier, 2014.

WOOLDRIDGE, M. *An Introduction to MultiAgent Systems*. [S.l.]: Wiley, 2009. ISBN 9780470519462.

XU, X.; CHEN, Y.; CALERO, J. M. A. 5 - Distributed decentralized collaborative monitoring architecture for cloud infrastructures. *Cluster Computing*, Springer US, p. 1 – 13, 2016. ISSN 15737543.

ZAINELABDEN, A. A. et al. On Service Level Agreement Assurance in Cloud Computing Data Centers. In: IEEE, 2016. *Cloud Computing (CLOUD), 2016 IEEE 9th International Conference on*. [S.l.], 2016. p. 921 – 926.

ZHANG, J.; LIU, Q.; CHEN, J. A Multi-agent Based Load Balancing Framework in Cloud Environment. In: IEEE, 2016. *Computational Intelligence and Design (ISCID), 2016 9th International Symposium on*. [S.l.], 2016. v. 1, p. 278 – 281.

ZIAFAT, H.; BABAMIR, S. M. *10 - A method for the optimum selection of datacenters in geographically distributed clouds*. Springer US, 2017. ISSN 0920-8542. ISBN 1122701719995. Disponível em: <http://link.springer.com/10.1007/s11227-017-1999-5>.