Brena Maia Santos
Orientador: Prof. Dr. Francisco Airton Pereira da Silva

Avaliação de Desempenho de Contêineres sob Dois Aspectos: Planejamento de Arquiteturas e Armazenamento de Dados

Picos - PI
30 de Maio de 2019

Brena Maia Santos
Orientador: Prof. Dr. Francisco Airton Pereira da Silva

Avaliação de Desempenho de Contêineres sob Dois Aspectos: Planejamento de Arquiteturas e Armazenamento de Dados

Monografia submetida ao Curso de Bacharelado em Sistemas de Informação como requisito parcial para obtenção de grau de Bacharel em Sistemas de Informação.

Universidade Federal do Piauí
Campus Senador Helvídio Nunes de Barros
Bacharelado em Sistemas de Informação

Picos - PI
30 de Maio de 2019

FICHA CATALOGRÁFICA
Serviço de Processamento Técnico da Universidade Federal do Piauí
Biblioteca José Albano de Macêdo

S237a Santos, Brena Maia.
Avaliação de desempenho de contêineres sob dois aspectos: planejamento de arquiteturas e armazenamento de dados. / Brena Maia Santos. – Picos, PI, 2019.
45 f.
CD-ROM: 4 ¾ pol.

Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação) – Universidade Federal do Piauí, Picos, 2019.
Orientador(A): Prof. Dr. Francisco Airton Pereira da Silva.

1. Banco de Dados - Gerenciamento . 2. Docker. 3. Contêiner (Computação). I. Título.

CDD 005.74

AVALIAÇÃO DE DESEMPENHO DE CONTÊINERES SOB DOIS ASPECTOS: PLANEJAMENTO DE ARQUITETURAS E ARMAZENAMENTO DE DADOS

BRENA MAIA SANTOS

Monografia aprovada como exigência parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Data de Aprovação:

Picos – PI, 12 de junho de 2019

Francisco Airton P. da Silva
Prof. Francisco Airton Pereira da Silva

Glauber Dias Gonçalves
Prof. Glauber Dias Gonçalves

Iúre de Sousa Fé
Prof. Iúre de Sousa Fé

Agradecimentos

Agradeço a Deus por ter me abençoado com saúde e força, me permitido realizar este trabalho. Ao professor Dr. Francisco Airton, pela orientação, incentivo e conhecimento repassado. A todos que os demais professores, que foram muito importantes para a obtenção de todo o conhecimento que possuo. A toda a minha família, em especial a minha mãe Jandira, e meus irmãos Janaíra, Beatriz e Janilton Jr, por todo o apoio. Agradeço também ao meu pai Janilton, que mesmo não estando mais presente fisicamente, tenho certeza que cuida de mim onde quer que esteja. Aos meus amigos Matheus, Tomaz, Walef, Estevão, Milton, Davi, Samuel, Douglas, João, Paulo, Daniel, Raphael, Erick, Jederilson, Naara, Wellington e João Paulo, que me ajudaram e estiveram presentes no meu dia a dia durante os últimos anos. Ao meu namorado Carlos Eduardo, por estar ao meu lado e por todo o apoio que me deu. A todos que de forma direta ou indireta fizeram parte da minha formação.

Resumo

O conceito de virtualização computacional, apesar de antigo, se tornou lugar-comum na viabilização de aplicações que compartilham recursos computacionais na infraestrutura da nuvem. As máquinas virtuais (do inglês *virtual machines*, VMs) trouxeram benefícios como a possibilidade de realizar *live migration* para prover tolerância a falha e balanceamento de carga. Porém, apesar das vantagens, as VMs adicionam camadas extras de abstração, resultando em perda de eficiência. Como alternativa as VMs, atualmente os contêineres se apresentam como uma solução mais leve e com bom desempenho. O Docker é uma plataforma para contêineres das mais populares, que fornece uma maneira automatizada de hospedar aplicações. Este trabalho realiza duas avaliações de desempenho baseadas em análise númerica e de sensibilidade com o objetivo de analisar a utilização de recursos durante o planejamento de uma arquitetura, e identificar os fatores que mais impactam na eficiência de um Sistema de Gerenciamento de Banco de Dados (SGBD), utilizando Docker como contêiner.

Palavras-chaves: contêiner, docker, desempenho, SPN, DoE, SGBD.

Abstract

The concept of computational virtualization, although old, has become a commonplace in enabling applications that share computing resources in the cloud infrastructure. Virtual machines (VMs) have brought benefits such as the ability to perform live migration to provide fault tolerance and load balancing. However, despite the advantages, VMs add extra layers of abstraction, resulting in loss of efficiency. As an alternative to VMs, today's containers present themselves as a lighter, better-performing solution. Docker is a popular container platform that provides an automated way to host applications. This work performs two performance evaluations based on numerical and sensitivity analysis with the objective of analyzing the resource utilization of a planning architecture and identifying the factors that most impact the efficiency of a Database Management System (DBMS) , using Docker as a container.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

MCC	*Mobile Cloud Computing* (Computação Móvel na Núvem)
MEC	*Mobile Edge Computing* (Computação Móvel na Borda)
VM	*Virtual Machine* (Máquina Virtual)
MRT	*Medium Response Time* (Tempo Médio de Resposta)
PN	*Petri Net* (Rede de Petri)
SPN	*Stochastic Petri Net* (Rede de Petri Estocástica)
IoT	*Internet of Things* (Internet das Coisas)
SGBD	*Data Base Management System* (Sistema de Gerenciamento de Banco de Dados)
DoE	*Design of Experiments* (Planejamento de Experimentos)

Sumário

1 Introdução

A virtualização é uma tecnologia estabelecida e em expansão. Muitos aplicativos, sistemas de gerenciamento, modelos e simuladores foram criados para promover e melhorar seu desempenho. Algumas das principais vantagens da virtualização incluem flexibilidade, capacidade, poder de processamento, crescimento da demanda e eficiência energética. No entanto, recentemente esse paradigma vem mudando. Um novo modelo de virtualização a nível do sistema operacional que permite instâncias com vários espaços de usuário adquiriu apoio de grandes empresas. Esse modelo de virtualização é chamado de contêiner. Esta tecnologia é semelhante a Máquinas Virtuais (VMs), porém os contêineres não exigem uma camada de emulação para serem executados, tornando seu funcionamento computacionalmente mais leve que as VMs (DUA; RAJA; KAKADIA, 2014). Os contêineres são criados com os recursos e processos do Sistema Operacional anfitrião e tem a capacidade de criar um ambiente de execução que isola as aplicações de forma que elas entendam estar funcionando em Sistemas Operacionais(SO) independentes, quando na verdade estão compartilhando o núcleo do SO anfitrião com outros contêineres(SILVA, 2017). O Docker é uma plataforma para contêineres das mais populares, que fornece uma maneira automatizada de hospedar aplicações.

O Docker permite a criação de contêineres em menos de um segundo, eliminando a sobrecarga bem conhecida da inicialização de *hypervisors* (FINK, 2014). Esta plataforma vem sendo bastante utilizada para atender um novo paradigma arquitetural que é a Internet das Coisas (Internet of Things - IoT) (AMENTO et al., 2018; HSIEH et al., 2018; MORABITO, 2017; NIU et al., 2017). Sabemos que alguns trabalhos na área de IoT utilizam bancos de dados para armazenar informações, como (JIANG et al., 2014) que propõe uma estrutura de armazenamento de dados para IoT e (GUPTA et al., 2016) que coletam dados de sensores e os enviam para um servidor de banco de dados para que sejam analisados e mantidos de forma estática. Visto isso, durante o planejamento da arquitetura do sistema é importante considerar o gasto de recursos e realizar avaliações de desempenho dos contêineres junto a bancos de dados, como um modo de auxiliar na melhoria da qualidade de serviço de aplicações. Com relação ao planejamento da arquitetura, não são encontrados estudos com análises de desempenho aprofundadas, mas alguns trabalhos analisaram contêineres Docker com banco de dados relacionais, tais como (VELÁSQUEZ; MUNOZ-ARCENTALES; RODRIGUEZ, 2018; XAVIER et al., 2016), que se concentraram na avaliação do desempenho da Rede e Disk I/O, respectivamente. No entanto, esses trabalhos também não realizam uma análise de sensibilidade aprofundada, tornando necessária uma investigação que utilize diversos fatores e níveis de forma conjunta, pois dessa forma é possível extrair informações mais precisas.

1.1 Definição do Problema

Avaliar o desempenho dos contêineres é importante para auxiliar administradores de infraestruturas computacionais a adequarem as arquiteturas de acordo com suas necessidades. Apesar de já ser uma tecnologia bastante conhecida, até o momento não existem trabalhos que realizam análises aprofundadas sobre o desempenho de contêineres em aspectos de planejamento de arquiteturas e armazenamento de dados, pois em alguns casos possui alta complexidade e alto custo financeiro.

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo geral deste trabalho é avaliar o desempenho do contêiner Docker sob a perspectiva de planejamento de arquiteturas e armazenamento de dados.

1.2.2 Objetivos Específicos

1. Desenvolver um modelo de Rede de Petri Estocásticas para analisar o desempenho de arquiteturas MEC.

2. Identificar os fatores que mais impactam na eficiência de um Sistema de Gerenciamento de Banco de Dados (SGBD) sendo executado em contêineres.

1.3 Organização do trabalho

O restante desta monografia se organiza da seguinte forma: O capítulo 2 contém alguns conceitos necessários para o entendimento deste trabalho; O capítulo 3 apresenta uma avaliação de desempenho baseada em Redes de Petri Estocásticas (SPN); o capítulo 4 apresenta uma avaliação de desempenho baseada em *Design of Experiments* (DOE); o capítulo 5 apresenta as principais conclusões obtidas acerca dos estudos realizados; e o capítulo 6 contém as publicações do autor deste trabalho.

2 Background

2.1 Computação Móvel na Borda

O surgimento de vários novos aplicativos de computação, como realidade virtual e ambientes inteligentes, tornou-se possível devido à disponibilidade de um grande pool de recursos e serviços em nuvem. No entanto, os aplicativos sensíveis a atraso apresentam alto nível de exigência para as infraestruturas computacionais convencionais. O paradigma da computação em nuvem não consegue atender aos requisitos de baixa latência, reconhecimento de local e suporte a mobilidade. Nesse contexto, o Mobile Edge Computing (MEC) foi introduzido para aproximar os serviços e recursos em nuvem aos usuários, aproveitando os recursos disponíveis nas redes de borda (AHMED; REHMANIB, 2017; WANG et al., 2017).

A MEC suporta diferentes opções de implementação, pois os servidores MEC podem ser localizados em diferentes locais dentro da rede de acesso de rádio, dependendo dos requisitos técnicos e de negócios (HU et al., 2015). A MEC vem se tornando viável para lidar com a grande geração de dados que ocorre na borda da rede e obter *insights* que auxiliem na tomada de decisões em tempo real, além de fornecer flexibilidade para usuários que desejam conservação de energia em baixa latência ou vice-versa no processamento de dados visuais (TRINH; YAO, 2017). Arquiteturas MEC tendem a ser mais responsivas do que as arquiteturas de Mobile Cloud Computing (MCC). Existem disparidades entre arquiteturas MEC e MCC em termos de servidor de computação, distância até os usuários finais e latência típica, etc. Tais características justificam a criação de novos modelos analíticos específicos para arquiteturas MEC.

2.2 Redes de Petri Estocásticas

Redes de Petri (*Petri Net - PN*) são uma família de formalismos baseados em estado, apropriada para modelar diversos tipos de sistemas que possuam mecanismos de concorrência, assincronicidade, distribuição, determinísticos, ou estocásticos. Sua representação gráfica permite a visualização de atividades concorrentes e dinâmicas do sistema. Enquanto suas características matemáticas tornam possíveis criar equações de estado, equações algébricas ou outros modelos matemáticos que regem o comportamento do sistema (MURATA, 1989).

A representação gráfica das PNs são formadas por: lugares (Figura 1 (a)) que correspondem às variáveis de estado; transições (Figura 1 (b)) que representam as ações e eventos do sistema; arcos (Figura 1 (c)) que presentam os fluxos de marcas pelo sistema;

e marcas (*tokens*) (1 (d)), cujo conjunto representa o estado do sistema em um instante. A realização de uma ação ou evento (disparo de uma transição) no sistema está ligado a pré-condições, deve existir uma relação entre os lugares e a transição para que esta possa ou não realizar a ação (disparar). Após o disparo alguns lugares terão suas informações alteradas (Marcas), ou seja, levará a uma pós-condição.

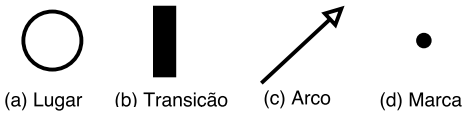

Figura 1: Elementos de uma Rede de Petri

Rede de Petri é uma técnica de modelagem que permite a representação de sistemas, utilizando como alicerce uma forte base matemática. Essa técnica possui a particularidade de permitir modelar sistemas paralelos, concorrentes, assíncronos e não-determiníticos. As Rede de Petri Estocásticas SPN são uma extensão das redes de Petri. As SPNs permitiram ligar as PNs a avaliação de desempenho, uma área normalmente baseada na abordagem de modelagem estocástica (MOLLOY, 1981).

Nas SPNs, as atividades que têm tempos associados, são representadas por transições temporizadas, que são representadas por retângulos brancos (Figura 2 (a)). O período de habilitação da transição corresponde ao tempo para realizar a atividade, e o disparo ao fim da atividade. Outra adição posterior das PNs, são os arcos inibidores (Figura 2 (b)), essa adição permite testar se um lugar não possui *tokens*. Com a presença do arco a transição estará habilitada se a quantidade de *tokens* no lugar p associado ao arco for menor que o peso do arco n, ou seja $M(p) < n$ (MARSAN et al., 1994).

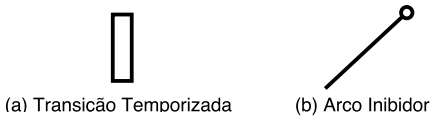

Figura 2: Elementos adicionais em uma SPN

Tanto a modelagem visual, como a obtenção das métricas por análise numérica e análise estacionária podem ser realizadas por ferramentas de apoio como a ferramenta Mercury (SILVA et al., 2015). O presente trabalho utilizou tal ferramenta para modelar e executar as análises numéricas.

2.3 *Design of Experiments*

O Desing of Experiments (DoE -Projeto de Experimentos) é uma metodologia estatística que foi aplicada no presente trabalho. O DoE foi amplamente utilizado com sucesso

em vários campos, tais como otimização (GUNST, 1996), química (LAZIC, 2006) e engenharia de software (KUHN; REILLY, 2002). Existe uma quantidade considerável de livros sobre tal metodologia (SIEBERTZ; BEBBER; HOCHKIRCHEN, 2017; SELTMAN, 2012; ANTONY, 2006); Portanto, a metodologia geral aqui será descrita brevemente. Em cada parte da aplicação, será exemplificado cada caso metodológico para o estudo de caso.

O conceito geral de DoE foi criado por uma série de experimentos reais simulados em um sistema ou modelo de sistema sob observação. Em cada experimento, um ou vários parâmetros de projeto são alternados e o impacto no comportamento do sistema é avaliado. Quais parâmetros são alterados e como eles são alterados é definido usando um plano de experimento. O objetivo é obter o máximo de informações possíveis usando a menor quantidade de experimentos. O comportamento do sistema com base nas mudanças de parâmetros é observado usando conjuntos de saídas. No contexto do DoE, as saídas podem ser referenciadas como "indicadores de performance", os parâmetros de projeto como "fatores" e os valores das configurações de "níveis".

Para analisar o impacto de cada fator no sistema e a interação com os outros fatores, várias combinações de fatores precisam ser testadas e, portanto, exige um experimento. Devido à grande quantidade de combinações possíveis, este é um esforço muitas vezes inviável em termos de tempo e custos. O DoE oferece uma coleção de métodos - referidos como planos experimentais ou tabelas de design - para reduzir a quantidade de experimentos necessários para encontrar informações precisas com o menor número de experimentos possível. O tipo de plano usado depende do objetivo do experimento. Encontrar a tabela de projeto ideal, por exemplo, a menor quantidade de experimentos necessários para descrever o comportamento dos sistemas corretamente, tem sido objeto de intensa pesquisa. Usando um conjunto de métodos estatísticos de avaliação nos dados resultantes do experimento, o impacto, efeitos e interações dos fatores em relação aos indicadores de performance escolhidos é avaliado. Os resultados do experimento podem ser usados para formular um modelo substituto matemático, também chamado de metamodelo em disciplinas de engenharia (MILLER et al., 2014; THOMAS et al., 2014). Neste trabalho foi aplicada a metodologia DoE para avaliação da arquitetura proposta. Três tipos de gráficos são adotados usualmente em estudos com DoE: Gráfico de Pareto, Gráfico de Efeitos Principais e Gráfico de Interação (ESPEJO, 2006).

2.3.1 Gráfico de Pareto

O gráfico de Pareto permite detectar qual o efeito da interação de fatores é mais importante para o processo ou estudo de otimização de projeto com o qual se deve lidar. Exibe os valores absolutos dos efeitos e desenha uma linha de referência no gráfico. Qualquer efeito que ultrapasse essa linha de referência é potencialmente importante. Um gráfico de Pareto é construído como na Figura 3, por exemplo. O gráfico mostra que os fatores B (tool geometry) e C (cutting angle) e a interação AC possui maior impacto.

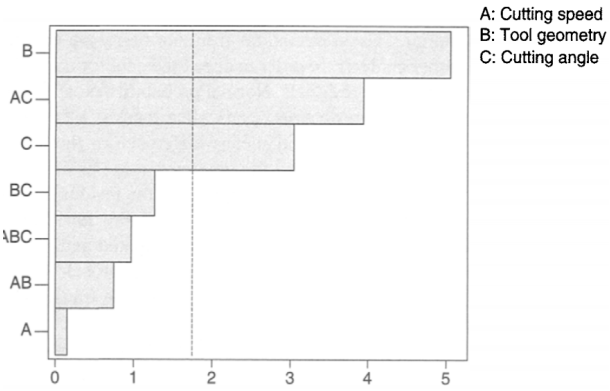

Figura 3: Gráfico de Pareto com Efeitos Padronizados

2.3.2 Gráfico de Efeitos Principais

O "Gráfico de Efeitos Principais" é o gráfico dos valores médios de resposta em cada nível de um parâmetro de projeto ou variável de processo. Pode-se usar esse gráfico para comparar a força relativa dos efeitos de vários fatores. O sinal e a magnitude de um efeito principal nos diriam o seguinte:

- O sinal de um efeito principal nos diz a direção do efeito, ou seja, se o valor médio da resposta aumenta ou diminui.
- A magnitude nos diz a força do efeito.

Se o efeito do parâmetro de projeto ou processo é positivo, isso implica que a resposta média é maior em nível alto do que em nível baixo da configuração do parâmetro. Em contraste, se o efeito for negativo, isso significa que a resposta média em nível baixo de configuração do parâmetro é maior que em nível alto. A Figura 4 ilustra o efeito principal da temperatura na resistência à tração de uma amostra de aço. Como pode ser visto na Figura, a resistência aumenta quando as configurações da temperatura variam para o nível baixo (ou seja, 1 para 1). O efeito de um parâmetro de projeto ou processo (ou fator) pode ser calculado matematicamente usando a simples Equação 2.1:

$$E_f = \overline{F}_{(+1)} - \overline{F}_{(-1)} \tag{2.1}$$

Quando $\overline{F}_{(+1)}$ significa a resposta média na configuração de alto nível de um fator, e $\overline{F}_{(-1)}$ significa a resposta média na configuração de baixo nível de um fator.

2.3.3 Gráfico de Interação

O processo de "Interações", identifica efeitos importantes e determina sua magnitude, logo, as interações entre os efeitos são cruciais. As interações ocorrem quando o efeito de um fator depende do nível de outro fator. Uma medida de design sempre aborda vários fatores. Entender como esses fatores interagem em que magnitude permite para escolher a

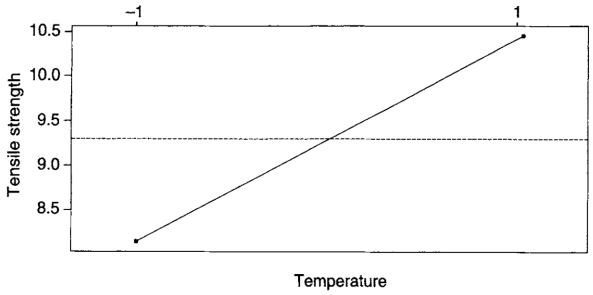

Figura 4: Gráfico de Efeitos Principais da Temperatura na Resistência à Tração

melhor combinação de medidas, revelando combinações de fatores com efeito cumulativo ou degradante. A interação entre os fatores A e B pode ser calculada usando a Equação 2.2.

$$I_{A,B} = \frac{1}{2}(E_{A,B(+1)} - E_{A,B(-1)}) \tag{2.2}$$

O $E_{A,B(+1)}$ é o efeito do fator 'A' no nível alto do fator 'B' e $E_{A,B(-1)}$ é o efeito do fator 'A' no nível baixo do fator 'B'.

Para determinar se dois parâmetros de processo estão interagindo ou não, pode-se usar uma ferramenta gráfica simples, porém poderosa, chamada de gráficos de interação. Se as linhas no gráfico de interação forem paralelas, não haverá interação entre os parâmetros do processo. Isso implica que a mudança na resposta média do fator 'A' não depende dos níveis do fator 'B'. Por outro lado, se as linhas não são paralelas, existe uma interação entre os fatores. Quanto maior o grau de afastamento de ser paralelo, mais forte o efeito de interação. Para interação sinérgica, as linhas no gráfico não se cruzam. Por exemplo, a Figura 5(a) é um exemplo de interação sinérgica.

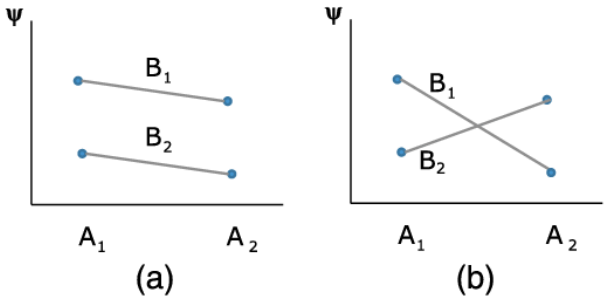

Figura 5: Gráfico de Interações Sinérgica e Antagônica

Na interação antagônica, as linhas no gráfico se cruzam. Isso pode ser ilustrado na Figura 5(b). Neste caso, a mudança na resposta média para o fator A no nível 'A_1' é alta em comparação ao nível 'A_2'. As mudanças nos níveis do fator 'A' para a resposta média, indica uma dependência do fator 'A' em relação aos níveis do fator 'B'.

3 Avaliação de Desempenho de Computação Móvel na Borda Usando Redes de Petri Estocásticas

De acordo com o relatório apresentado pela Cisco Visual Networking Index, o número de dispositivos móveis chegará a 11,6 bilhões em 2020 (JUNG, 2016). A tendência de aumento no uso de dispositivos móveis é fundamentalmente impulsionada pelo aumento de usuários móveis e pelo desenvolvimento de aplicativos cada vez mais interativos e utilitários (KITANOV; MONTEIRO; JANEVSKI, 2016) (BECK et al., 2014). O crescimento do uso de smartphones e tablets teve um efeito extremo sobre as redes móveis trazendo grandes desafios para as companhias de telecomunicações (CAU et al., 2016).

As redes celulares devem suportar a todo custo a baixa capacidade de armazenamento, o alto consumo de energia, a baixa largura de banda e altas latências (ORSINI; BADE; LAMERSDORF, 2015). Com o objetivo de propor arquiteturas cada vez mais otimizadas para este contexto, surgiu a computação móvel na borda, do inglês *Mobile Edge Computing* (MEC). O principal objetivo da MEC é implantar os recursos ainda mais perto dos usuários — na borda da rede. Assim, a computação e armazenamento são executados mais próximos do local de origem, sempre na casa dos milissegundos (JARARWEH et al., 2016b). Por ser uma área ainda recente e com grandes desafios, algumas lacunas de pesquisa ainda devem ser exploradas, como é o caso da avaliação de desempenho de arquiteturas MEC.

Este capítulo apresenta um modelo SPN para modelar uma arquitetura MEC onde os recursos de um único servidor são paralelizados com microsserviços sendo executados em múltiplos contêineres. O modelo proposto neste trabalho permite avaliar o *trade-off* entre tempo médio de execução (MRT) e utilização de recursos. Em resumo, as principais contribuições deste estudo são:

- um modelo analítico, que é uma ferramenta útil para administradores de sistemas avaliarem o desempenho de arquiteturas MEC, antes mesmo que sejam implantadas;
- um conjunto de análises numéricas com dados reais que provê um guia prático para análise de desempenho em arquiteturas MEC.

3.1 Trabalhos Relacionados

Alguns trabalhos relacionados propõem arquiteturas MEC em **cooperação com outras camadas**. No entanto, esses trabalhos não exploram diferentes configurações de re-

cursos computacionais. (TONG; LI; GAO, 2016) propõem implantar servidores de nuvem na borda da rede e posicionar os servidores de forma hierárquica distribuídos geograficamente, de modo a utilizar a nuvem para atender às cargas de pico de requisições advindos da MEC. (CHEN et al., 2016) foca em um algoritmo de otimização no qual calcula onde executará a carga de trabalho (MEC ou nuvem). Os autores utilizam como validação aplicações genéricas que necessitam de alto nível de processamento. (LI; WU, 2018) foca na capacidade da MEC e da nuvem em processar grandes cargas de dados. O objetivo principal do trabalho é tentar diminuir a latência entre as duas camadas atribuindo pesos às complexidades das tarefas a serem executadas remotamente.

Alguns trabalhos focam no problema do **consumo de energia de dispositivos móveis** com auxílio de uma arquitetura MEC. (KE; S; L, 2016) estudou mecanismos de *offloading* investigando uma arquitetura MEC e redes heterogêneas 5G. Os autores formularam um problema de otimização para minimizar o consumo de energia observando tempo de processamento e transferência de dados. (TRINH; YAO, 2017) fez um estudo sobre o potencial da MEC para mitigar a limitação de bateria em dispositivos IoT. Os autores utilizaram uma aplicação de reconhecimento facial para demonstrar a viabilidade das políticas de decisão de *offloading*. (MAO; ZHANG; LETAIEF, 2016) também propôs um algoritmo de otimização de bateria em dispositivos móveis. O algoritmo incluiu freqüências do ciclo da CPU do servidor MEC e taxa de latência. Uma vantagem desse algoritmo é que as decisões dependem apenas das informações instantâneas do lado do servidor sem exigir informações de distribuição da solicitação de tarefa. Diferentemente desses trabalhos, este estudo não analisa diretamente o gasto energético dos dispositivos clientes, no entanto a métrica MRT aqui adotada está proporcionalmente relacionada ao gasto energético dos dispositivos clientes. Quanto mais tempo a requisição demorar para executar maior será o consumo energético.

Os trabalhos mais relacionados ao presente trabalho tratam sobre o **planejamento da infraestrutura MEC**. (PREMSANKAR; TALEB, 2018) realizou uma avaliação experimental real com uma aplicação de um jogo eletrônico de alta interatividade, porém com a limitação de haver apenas um cliente móvel. O trabalho de (JARARWEH et al., 2016a) desenvolveu um simulador de uma arquitetura que incluía uma camada de Cloudlet e uma camada MEC visando aumentar a área de cobertura para os usuários móveis, onde os usuários podem realizar requisições com custos mínimos em termos de gasto energético e tempo de execução. Uma limitação é que os autores adotam apenas o número de requisições como parâmetro de entrada e também o fato de usar um simulador próprio sem explicações detalhadas. (LIU et al., 2016) adotaram um modelo de cadeias de Markov como processo de decisão sobre onde as tarefas devem ser executadas (localmente ou no servidor MEC). O modelo leva em consideração o estado de enfileiramento do *buffer* de tarefas, o estado de execução do dispositivo móvel e o estado da rede. No entanto os autores não consideram o servidor MEC com múltiplos nós paralelos, e assim, não aproveitando

substancialmente do potencial do paralelismo do servidor. (BADRI et al., 2017) também utilizou cadeias de Markov, porém, com o objetivo de decidir onde executar requisições em múltiplas torres de comunicação MEC. O algoritmo leva em conta a movimentação dos usuários, o custo de comunicação entre usuários e servidores, o custo de execução em cada servidor e o custo de realocações. Este trabalho não foca em múltiplos servidores e sim no planejamento de uma infraestrutura mínima com um único servidor MEC com paralelismo através de múltiplos contêineres.

Diferente da nossa proposta, os trabalhos acima citados não exploram nível de utilização de recursos e apenas alguns consideram MRT. Todos os trabalhos são limitados em termos de parametrização das avaliações, com no máximo observando três parâmetros de configuração da arquitetura. Nenhum dos trabalhos (com exceção de (PREMSANKAR; TALEB, 2018)) abordou aplicações com alto nível de interação. Ademais, nenhum dos trabalhos adotou Redes de Petri Estocásticas.

3.2 Arquitetura Base

A Figura 6 ilustra a arquitetura que propomos modelar e analisar seu desempenho. Tal arquitetura base é bastante adotada para descrever a infraestrutura MEC em diversos trabalhos (TRINH; YAO, 2017; YUANZHE; SHANGGUANG, 2018; LI; WU, 2018) com um servidor MEC visando processar fluxos de dados recebidos. Esses dados são gerados e enviados por aplicativos executados em dispositivos móveis, por exemplo, dados de aplicações de monitoramento da saúde do usuário, dados para a renderização de jogos, etc. No contexto deste trabalho focamos em aplicações com alto nível de interatividade com o usuário, incluindo periféricos como smartphones ou tablets.

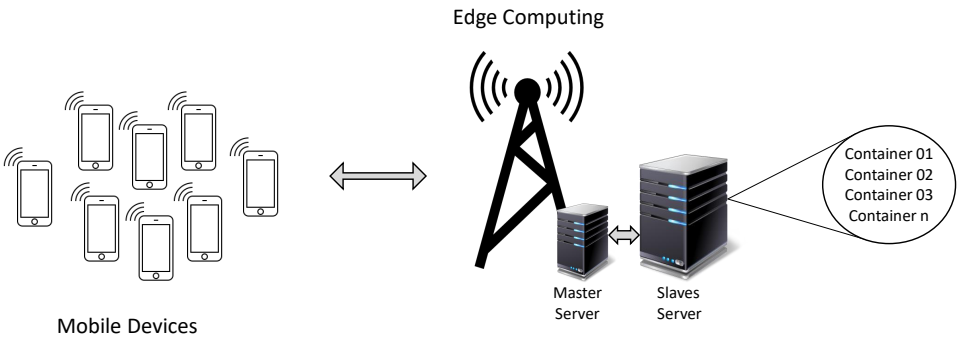

Figura 6: Arquitetura MEC Base Adotada na Avaliação de Desempenho

Mais detalhadamente, na camada da *edge computing* temos dois servidores, um servidor *master* e um servidor com os nós *slaves* (que farão o processamento em si). Os nós *slaves* são microsserviços que são executados em contêineres. Neste trabalho, cada contêiner é configurado para executar em um núcleo do servidor. Portanto, se existirem 16

núcleos, serão executados 16 contêineres. O uso de contêineres na área de MEC ainda não é muito comum. O uso de contêineres permite maior flexibilidade para escalar o poder computacional da arquitetura de acordo com o volume de tarefas e também restrições de tempo de resposta de aplicações.

O servidor master é responsável por receber as requisições dos dispositivos móveis e distribuí-los entre os nós slaves. A princípio, o servidor master executa o serviço de gerenciamento com o serviço sendo executado em modo bare metal (sem virtualização) e com uso de threads. Porém, nada impede de também virtualizar o serviço do servidor master. Para gerenciar os contêineres consideramos que um orquestrador (ex.: Kubernetes[1] ou Swarm[2]) de contêineres deve ser utilizado visando maior confiabilidade e elasticidade dos recursos.

Como ilustra a Figura 7, existem 3 intervalos de tempo no fluxo de comunicação da arquitetura: (i) Arrival Delay (AD): intervalo de tempo entre as gerações das requisições; (ii) Distribution Delay (DD): intervalo de tempo para os trabalhos serem distribuídos ou encaminhados aos nós; e (iii) Processing Delay (PD): tempo de processamento das requisições pelos nós.

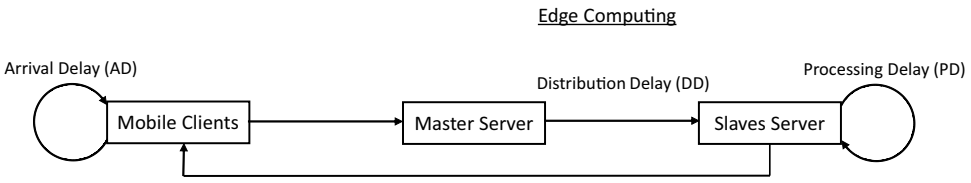

Figura 7: Intervalos de Tempo na Comunicação dos Componentes da Arquitetura

3.3 Proposta de Modelo SPN

Nessa seção, descrevemos a nossa proposta de modelo baseado em redes de Petri estocásticas (SPN) para representar a arquitetura que integra módulos na borda da rede apresentada na seção anterior. Ressaltamos que o objetivo desse modelo é possibilitar a checagem do desempenho de mudanças no sistema, antes mesmo que elas sejam implantadas. A Figura 13 apresenta o modelo SPN, composto de duas macro partes:

1. *Admission* que trata da geração das requisições;
2. *Edge* composto pelo servidor master e o servidor com nós slaves. O servidor master recebe dados e os distribui entre os nós slaves, que por fim retornam os resultados aos clientes;

[1] Kubernetes: https://kubernetes.io/
[2] Swarm: https://docs.docker.com/engine/swarm/

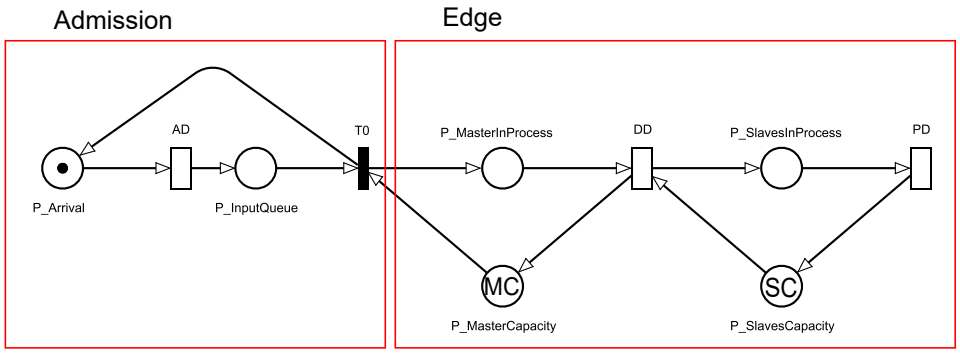

Figura 8: Modelo SPN para uma Arquitetura Edge Computing

Os componentes são representados por elementos gráficos do tipo lugares (círculos), transição (barra preenchida), transições temporizadas (barras vazias) e marcações de lugares (círculos preenchidos). A Tabela 1 apresenta todos os elementos do modelo extensivamente. As transições temporizadas são parametrizadas com distribuições de probabilidade. O administrador do sistema deve informar essas distribuições de acordo com a literatura ou realizando medições e caracterizações do sistema.

Tabela 1: Descrição dos Elementos do Modelo

Tipo	**Elemento**	**Descrição**
Lugares	P_Arrival	Espera por novas requisições
	P_InputQueue	Espera pela disponibilidade da fila
	P_MasterInProcess	Trabalhos na fila do servidor master
	P_MasterCapacity	Capacidade do servidor master
	P_SlavesInProcess	Requisições na fila dos nós slaves da borda
	P_SlavesCapacity	Capacidade da borda
Transições Temporizadas	AD	Tempo de chegada entre requisições (*Single Server*)
	DD	Tempo para o servidor master distribuir as requisições entre os nós slaves (*Infinite Server*)
	PD	Tempo gasto para processar o trabalho no nó slave(*Infinite Server*)
Marcações dos Lugares	MC	Capacidade máxima da fila de entrada do master
	SC	Capacidade de nós slaves disponíveis para processamento

Dada a visão geral do modelo, descrevemos agora o fluxo de processamento de dados entre seus componentes. A sub-rede *Admission* é composta por dois lugares **P_Arrival** e **P_InputQueue**, que representam a espera entre gerações de requisições e a aceitação dessas requisições na fila, respectivamente. Os tokens em **P_Arrival** e **P_InputQueue** representam qualquer tipo de requisição que envolva entrada de dados. Os tempos entre chegadas de requisições é atribuído à transição **AD**. Consideramos que os tempos entre disparos são exponencialmente distribuídos, essa suposição pode ser modificada, alterando essa distribuição. A transição **AD** não leva em conta as perdas provenientes da rede.

Quando **T0** é disparado, a sub-rede *Edge* é alcançada. A quantidade de tokens em

P_MasterInProcess representa o enfileiramento de requisições na Edge. A marcação **MC** em **P_MasterCapacity** indica a quantidade de espaço de armazenamento temporário do servidor master, enfileirando as requisições. O enfileiramento ocorre quando não há a capacidade disponível para servir a requisição recém-chegada. Assim, caso haja espaço de armazenamento suficiente, um token é retirado de **P_InputQueue** e de **P_MasterCapacity**, alocando um token a **P_MasterInProcess**. Quando isto acontece, o lugar da entrada na subrede *Submission* (**P_Arrival**) é então habilitada, permitindo um novo disparo.

O disparo de **DD** representa o início da distribuição de requisições aos nós slaves. Estes disparos são condicionados à quantidade de nós disponíveis para processamento em **P_SlavesCapacity** (com marcação **SC**). A marcação **SC** indica a quantidade de nós disponíveis na borda da rede. À medida que as requisições vão sendo consumidas pelo lugar **P_SlavesInProcess**, os tokens são retirados do **P_SlavesCapacity**. Este fluxo significa que cada trabalho será alocado em um recurso à medida que for chegando.

PD representa o tempo gasto pelo nó slave para processar uma requisição. Quando **PD** é disparada, um token é retirado de **P_SlavesInProcess** e um token é retornado a **P_SlavesCapacity**. A transição AD possui semântica *single server* de uma distribuição exponencial, pois estamos considerando taxas de chegada exponencialmente distribuídas. Todas as outras transições possuem a semântica *infinite server*, então, cada trabalho é processado independentemente. É importante notar que o tempo de processamento depende muito da capacidade computacional de cada nó. Neste trabalho consideramos que todos os nós de cada camada possuem a mesma capacidade computacional.

O modelo proposto permite avaliar um número muito grande de cenários, pois o avaliador deve configurar 5 parâmetros (vide Tabela 1). Os parâmetros incluem: as 3 transições temporizadas, e as 2 marcações de lugares relacionados a recursos ou carga de trabalho. Qualquer alteração em um destes parâmetros pode impactar significativamente no tempo médio de resposta do sistema e no nível de utilização dos recursos. A variação das possibilidades de cenários considerando um grande número de fatores é o que torna este modelo a principal contribuição deste trabalho.

3.4 Métricas de Desempenho

Nesta seção, definimos métricas para avaliar a arquitetura de borda com base no modelo proposto. O tempo médio de resposta (MRT) pode ser obtido a partir da Lei de Little (LITTLE, 1961). A Lei de Little relaciona o número médio de requisições em progresso em um sistema ($RequestsInProcess$), a taxa de chegada de novas requisições (ARR) e o tempo médio de resposta (MRT). A taxa de chegada é o inverso do tempo de chegada. Considerando a transição para tempo entre gerações do modelo, temos que $ARR = \frac{1}{AD}$. Vale ressaltar que a Lei de Little requer um sistema estável, ou seja, que

possua uma taxa de requisições menor que a taxa de processamento dos servidores. No modelo proposto pressupomos que a taxa de chegada real não necessariamente é a taxa de chegada efetiva, pois alguns trabalhos podem se perder, ou por termos filas finitas, serem descartados. Então, como recomendado pelo autor Jain (1990), a probabilidade de descartes representado pela variável $N\_Discard$ é subtraída. Portanto, a equação correspondente à Lei de Little para MRT utilizada no modelo proposto é expressa na Equação 3.1.

$$\textbf{MRT} = \frac{RequestsInProgress}{ARR \times (1 - N\_Discard)} \tag{3.1}$$

A Equação 3.2 define $RequestsInProcess$. Para calcular o número de requisições em progresso no sistema, precisa-se somar a quantidade de tokens em cada um dos lugares que representam uma requisição em andamento. Na Equação 3.2, $Esp(Lugar)$ representa a esperança estatística de existir tokens em “Lugar", onde $Esp(Lugar) = (\sum_{i=1}^{n} P(m(Lugar) = i) \times i)$. Em outras palavras, $Esp(Lugar)$ indica qual a quantidade média esperada de tokens que ocupam aquele Lugar.

$$\textbf{RequestsInProcess} = Esp(P\_MasterInProcess) + Esp(P\_SlavesInProcess) \tag{3.2}$$

A equação 3.3 define $N\_Discard$. Para calcular o descarte é necessário existir token na fila de entrada (P_ArrivalQueue) e não restar mais nenhum recurso disponível dentro das sub-redes. $P(Lugar = n)$ calcula a probabilidade de existirem n tokens em “Lugar".

$$\textbf{N\_Discard} = P((P\_InputQueue = 1) \wedge (P\_MasterCapacity = 0) \wedge (P\_SlavesCapacity = 0) \tag{3.3}$$

Por fim, além do MRT nós calculamos também a probabilidade de utilização dos recursos. A utilização do master é dada pela Equação 3.4, e a utilização dos nós slaves é calculada pela Equação 3.5. A utilização é obtida dividindo o número de tokens do lugar correspondo aos elementos em uso dividido pela capacidade total daqueles recursos.

$$\textbf{U\_Master} = \frac{Esp(P\_MasterInProcess)}{MC} \tag{3.4}$$

$$\textbf{U\_Slaves} = \frac{Esp(P\_SlavesInProcess)}{SC} \tag{3.5}$$

3.5 Análises Numéricas

Esta seção apresenta duas análises numéricas a partir da resolução do modelo SPN proposto visando analisar MRT e Utilização. No trabalho de (PREMSANKAR; TALEB, 2018) (que iremos chamar de “trabalho de referência") os autores avaliaram uma arquitetura MEC com um único dispositivo móvel como cliente através de experimentos em um ambiente real com execução de serviços em contêineres. O trabalho de referência avaliou

um jogo 3D chamado Neverball onde o jogador deve inclinar o piso para controlar a bola fazendo com que ela colete moedas e alcance um ponto de saída antes que o tempo acabe. Extraímos do trabalho de referência os parâmetros de entrada do modelo. Portanto, este estudo evolui o trabalho dos autores pois nós agora analisamos numericamente cenários com múltiplos usuários considerando o jogo com uma resolução de 800x600 pixels. O parâmetro adotado do trabalho de referência correspondendo ao tempo de processamento (PD) de uma requição foi de 24ms. Adotamos o tempo de 5ms para a distribuição das requisições (transição DD). Neste estudo estabelecemos que o servidor master tem uma restrição de atender no máximo 40 requisições paralelamente, ou seja setamos a marcação MC com valor 40.

O modelo permite uma ampla variedade de parametrizações. Na presente análise variamos dois parâmetros, o intervalo de tempo entre chegadas de requisições (AD) e a capacidade de recursos do servidor com nós slaves (SC). O valor de AD foi variado entre 1ms até 10ms com acréscimos de 0.5ms. A variável SC foi configurada com três possibilidades (8, 16 e 32), correspondendo a opções sobre a quantidade de núcleos de um servidor. Obviamente todos estes parâmetros poderiam ser variados de outras formas, por exemplo, o número de nós slaves poderiam não estar atrelados a número de núcleos, podendo adotar por exemplo SC na casa dos milhares. Adotando os parâmetros acima mencionados, a seguir apresentamos os resultados considerando as métricas MRT, descarte, utilização do servidor master, e utilização do servidor com nós slaves.

A Figura 9 apresenta os resultados para o tempo médio de resposta. A princípio espera-se que quanto maior o intervalo de tempo entre chegadas (AD) menor seja o valor de MRT, pois o sistema estará mais ocioso com menos chegadas de requisições. Espera-se também que quanto maior for a quantidade de recursos slaves (SC) menor será o valor de MRT pois se aumentará a disponibilidade de recursos, aumentando-se o nível de paralelismo. Estes dois comportamentos são facilmente observados nos resultados quando os valores de descartes de trabalhos são desprezíveis no modelo (que podem ser observados na Figura 12), para esses intervalos o MRT decresce até o tempo mínimo para executar trabalhos sem enfileiramento no sistema.

No entanto, para os casos que o descarte é presente no sistema, observamos o aumento do MRT até um pico para que ele decresça, este comportamento é decorrente do valor limitante de recursos no sistema, onde parte dos trabalhos entrantes são descartados quando não há mais recursos, limitando a variação do MRT ao tempo entre chegadas, como pode ser deduzido da lei de Little (JAIN, 1990), portanto o tempo médio entre saídas irá aumentar junto com o tempo médio entre chegadas até o pico, para SC = 8 foi em AD = 1.5ms e para SC = 16 foi de AD = 3.0ms. Ao atingir esses tempos entre chegadas, a quantidade de trabalhos dentro do sistema começa a reduzir drasticamente, reduzindo o MRT mesmo com o aumento do AD. É importante ressaltar que consideramos no MRT a taxa de chegada efetiva, isto é, ajustando o seu valor com a probabilidade de

descarte.

O valor de MRT para SC = 32 é extremamente baixo, mesmo para um intervalo entre chegadas de 1ms. Comparando SC = 16 e SC = 32, temos que a partir de 2.5ms os MRTs se igualam, e a partir de 5.5ms a configuração SC = 8 também apresenta o mesmo resultado médio. Portanto, caso o contexto real apresentasse um AD de 5.5ms, um servidor de 8 núcleos atingiria o mesmo desempenho do que servidores mais potentes, portanto este trabalho pode auxiliar no dimensionamento adequado da escolha de servidores, identificando a melhor opção de desempenho e custo para uma carga de trabalho esperada.

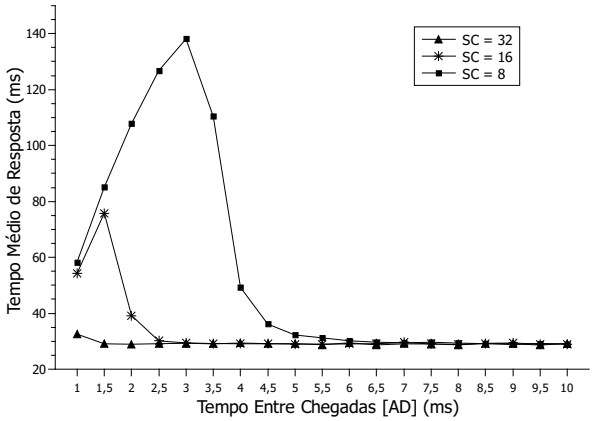

Figura 9: Tempo Médio de Resposta (MRT)

A Figura 10 apresenta o nível de utilização do servidor master. O servidor master é o primeiro componente que a requisição atinge ao entrar na camada MEC. Para as configurações SC = 8 e SC = 16, o nível de utilização gira em torno de 100% nos menores valores de AD, posteriormente este valor cai. Para SC = 32, mesmo com AD = 1.0ms, o valor de utilização chega a apenas 20%. A partir de AD = 5.5ms as três configurações possuem valores semelhantes e próximos a 0%. O administrador do sistema deve considerar se deseja níveis altos ou baixos de ociosidade para seu servidor master. A Figura 11 mostra o nível de utilização do servidor com nós slaves. Quanto maior o número de recursos, menor o nível de utilização dos nós slaves. À medida que aumenta-se AD, o nível de utilização decai de forma sutil nos três casos. No entanto, esta queda somente inicia-se em AD = 3.0ms para SC = 8 e em AD = 1.5ms para SC = 16. Até estes pontos, o nível de utilização gira em torno de 82%, o que causa o comportamento do MRT explicado anteriormente.

A Figura 12 apresenta a probabilidade de descarte de novas requisições. Para SC = 32 a probabilidade de descarte é igual a zero. Portanto, caso seja possível adquirir um servidor com 32 núcleos não haverá descarte independente do intervalo entre chegadas de requisições. Para SC = 8 e SC = 16 somente a partir de AD = 2.0ms e AD = 4.0 as probabilidades de descarte tendem a zero. Estes intervalos iniciais com descarte estão diretamente relacionados ao alto nível de utilização apresentados por ambos os servidores,

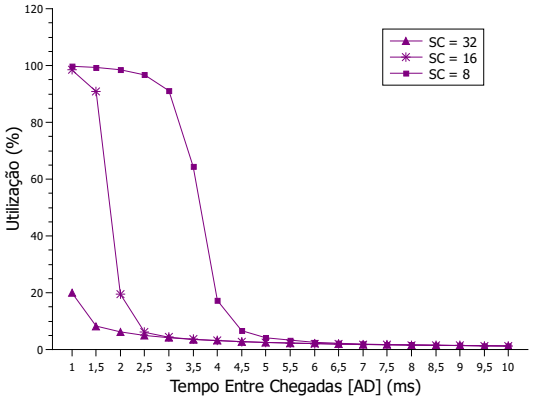

Figura 10: Utilização do Servidor Master

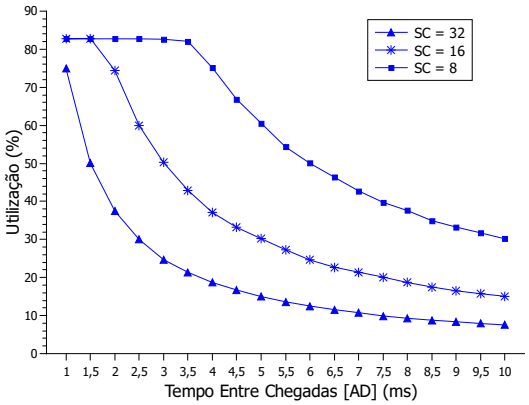

Figura 11: Utilização do Servidor Slave

impactando diretamente no tempo médio de resposta do sistema. Portanto, qualquer análise estocástica realizada com o modelo proposto deve observar as quatro métricas em questão para obter uma visão completa do comportamento do sistema. Também é possível identificar os limites de funcionamento do sistema, ou seja, até quantos trabalhos podem ser perdidos sem que comprometa a utilidade do sistema.

3.6 Resumo do Capítulo

A Computação Móvel na Borda (MEC) surgiu como uma alternativa para diminuir a latência da rede, fazendo com que o processamento do fluxo de dados ocorresse mais próximo dos usuários mobile. No entanto, a configuração dos servidores pode influenciar diretamente no desempenho da arquitetura MEC. Este capítulo propôs um modelo de Rede de Petri Estocástica (SPN) para modelar tal cenário e analisar seu desempenho, considerando diversos parâmetros que podem afetar diretamente no Tempo Médio de Resposta (MRT) e no Nível de Utilização. O capítulo apresenta também análises numéricas com valores de entrada reais que servem como um guia prático para auxiliar adminis-

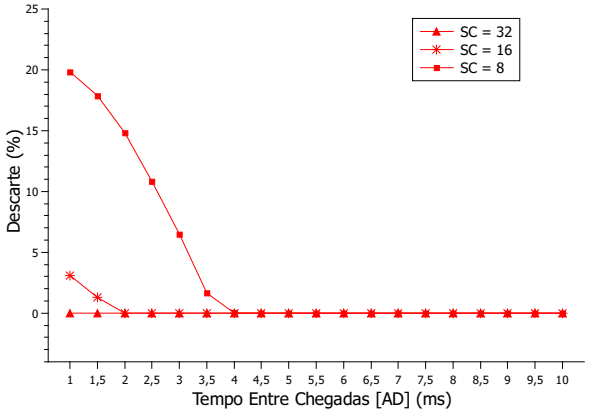

Figura 12: Probabilidade de Descarte

tradores de infraestruturas computacionais a adequar suas arquiteturas, encontrando o *trade-off* entre o MRT e o nível de utilização. A análise numérica permitiu ver que a taxa entre chegadas possui grande impacto sobre o sistema. Para valores muito pequenos observou-se níveis de descarte, o que pode comprometer o resultado das outras métricas, principalmente do MRT.

4 Uma Avaliação de Desempenho de Contêineres Docker Executando Diferentes SGBDs Relacionais

A plataforma Docker vem sendo bastante utilizada em Internet das Coisas (Internet of Things - IoT). Alguns trabalhos nessa área utilizam bancos de dados para armazenar informações, como (JIANG et al., 2014) e (GUPTA et al., 2016). Visto isso, é importante realizar avaliações de desempenho de contêineres junto a bancos de dados, como uma forma de auxiliar na melhoria da qualidade de serviço de aplicações. Alguns trabalhos analisaram contêineres Docker com banco de dados relacionais, mas não realizaram uma análise de sensibilidade profunda, tornando necessária uma avaliação com diversos fatores e níveis de forma conjunta, para extrair informações mais precisas.

Este capítulo apresenta uma análise sobre o uso de contêineres Docker executando banco de dados relacionais. Para tanto, foi realizada uma avaliação de desempenho baseada em análise de sensibilidade sobre o contêiner Docker em comparação com diferentes bancos de dados. Foram considerados os seguintes fatores: SGBD utilizado, número de tabelas e número de tuplas. E como variáveis dependentes, foram adotadas as seguintes: Tempo de Execução para processamento dos dados e Nível de Falha de requisições. Assim, este capítulo tem como principal contribuição o provimento de indicativos de qual banco de dados relacional deve ser adotado junto ao Docker visando um melhor desempenho.

4.1 Trabalhos Relacionados

Esta seção apresenta alguns trabalhos relacionados ao presente estudo, ou seja, avaliaram o desempenho do contêiner Docker em conjunto com algum SGBD.

Os autores em (MORABITO, 2016) avaliaram o uso de contêineres Docker em ambientes restritos. Os autores avaliaram por meio de diferentes ferramentas de benchmark, o desempenho do Docker quando executado em um computador de placa única, no caso o Raspberry Pi (RPi2). Os resultados mostram um impacto quase insignificante da camada de virtualização de contêiner em termos de desempenho, se comparado à execução nativa.

Os autores em (BARIK et al., 2016) avaliaram o desempenho de VMs e contêineres simultaneamente em computadores tradicionais. Os autores mostram que contêineres têm algumas deficiências nas áreas de segurança e isolamento, pois proporciona um isolamento de segurança mais baixo do que a virtualização.

Os autores em (VELÁSQUEZ; MUNOZ-ARCENTALES; RODRIGUEZ, 2018) apresentam uma análise dos diferentes sistemas de banco de dados para armazenar as informa-

ções coletadas por uma rede de sensores sem fio. Diferentes testes são apresentados para avaliar SGBDs relacionais (MySQL, PostgreSQL) e No-SQL (MongoDB, Couch, Neo4J). Os SGBDs foram implantados em contêineres Docker. O objetivo do trabalho foi fornecer informações necessárias para a correta escolha da base de dados para esse tipo de sistema. Os resultados indicaram que o melhor SGBD analisado para a ingestão de dados por segundo é o PostgreSQL, mas, para a quantidade de armazenamento total, o MongoDB obteve melhor desempenho.

Os autores em (SHIRINBAB; LUNDBERG; CASALICCHIO, 2017) apresentam uma extensa comparação de desempenho entre o contêiner VMware e o Docker, enquanto executa o Apache Cassandra. O Apache Cassandra lidera o ranking de SGBD distribuído NoSQL quando se trata de plataformas de Big Data. Como linha de base para comparações, os autores avaliaram o Cassandra focando na infraestrutura física. O estudo mostrou que o Docker tinha uma sobrecarga menor em comparação com o VMware ao executar o Cassandra. O desempenho do Cassandra na infraestrutura com Docker era tão bom quanto no ambiente sem virtualização alguma.

Todos os trabalhos acima destacados utilizaram a ferramenta Docker como hospedeiro para executar alguma instância de SGBD. Pôde-se verificar várias conclusões comparativas entre bancos de dados relacionais e até não relacionais. No entanto, nenhum dos trabalhos observados realizaram uma análise de sensibilidade sobre fatores que interferem no desempenho do Docker junto ao SGBD. A Tabela 2 resume as comparações.

Tabela 2: Trabalhos Relacionados

Trabalho Relacionado	**Métricas**	**SGBD**	**Hardware**	**Análise de Sensibilidade**
(MORABITO, 2016)	CPU, Memória, Tipo de Transação	MySQL	Raspberry	Não
(BARIK et al., 2016)	Segurança	MySQL	Computador Desktop	Não
(SHIRINBAB; LUNDBERG; CASALICCHIO, 2017)	CPU, Tipo de Transação	Apache cassandra	Computador Desktop	Não
(VELÁSQUEZ; MUNOZ-ARCENTALES; RODRIGUEZ, 2018)	Segurança	MySQL, PostgreSQL, MongoDB, Couch, Neo4J	Computador Desktop	Não
O presente trabalho	Tempo de Execução, Falhas	MySQL, PostgreSQL, SQLServer	Computador Desktop	Sim

4.2 Análise de Sensibilidade do Contêiner Docker e SGBDs Relacionais

Esta seção descreve os experimentos realizados para analisar os fatores com maior influência no desempenho de contêineres com SGBDs relacionais, baseado em duas métricas: tempo de execução e nível de falha na requisição.

4.2.1 Desenho Experimental

O ambiente de experimentos foi composto por dois computadores conectados à mesma rede: o Computador A foi utilizado para hospedar o contêiner Docker, enquanto o Computador B foi utilizado para enviar as requisições de acesso aos dados presentes no contêiner. A Tabela 3 apresenta as configurações das respectivas máquinas.

Tabela 3: Configurações dos Computadores A e B

	Processador	Memória	Disco Rígido	Sistema Operacional
Computador A	Intel i5	8 GB	1 TB	Linux Ubuntu 18.04 LTS
Computador B	Intel i3	4 GB	500 GB	Linux Ubuntu 16.04 LTS

Para realizar a avaliação de desempenho, duas métricas foram definidas: tempo de execução e nível de falha na requisição. Para a avaliação da métrica de tempo de execução, foi realizado o *Design of Experiments* (DoE), com três fatores estabelecidos: SGBD, número de tabelas e número de tuplas da tabela. O SGBD possui três níveis, que foram escolhidos por serem alguns dos mais populares: MySQL (v. 8.0.12), PostgreSQL (v. 11.0) e SQL Server (v. 2017). Vale ressaltar que as configurações padrão dos SGBDs foram mantidas, e a consulta realizada buscava os dados de todas as colunas das tabelas (ID e NOME). O fator número de tabelas se refere à quantidade de tabelas que serão consultadas durante o experimento e possui dois níveis: 1 e 2. Já o fator número de tuplas é referente à quantidade de tuplas que cada tabela consultada possui e foram determinados três níveis: 1000, 3000 e 5000. A escolha do número de tuplas e tabelas foi baseada em sucessivos testes com níveis de fatores mínimos para garantir um controle sobre o test-bed. Nos experimentos em que são utilizadas duas tabelas, foi realizada a operação de junção, e o número de tuplas das mesmas devem ser iguais. A Tabela 4 apresenta os fatores e níveis escolhidos para a realização do DoE utilizando a métrica de tempo de execução.

Tabela 4: Fatores e níveis do DoE para a métrica tempo de execução

	Nível 1	Nível 2	Nível 3
SGBD	MySQL	PostgreSQL	SQL Server
Número de Tabelas	1	2	-
Número de Tuplas	1000	3000	5000

Com todos os fatores e níveis definidos, deve-se combinar os fatores e níveis para definir como os experimentos deverão ser realizados. A Tabela 5 apresenta as combinações possíveis entre os fatores e níveis.

Tabela 5: Combinações de fatores e níveis para a métrica tempo de execução

Nº	SGBD	Número de Tabelas	Número de Tuplas
1	MySQL	1	1000
2	MySQL	1	3000
3	MySQL	1	5000
4	MySQL	2	1000
5	MySQL	2	3000
6	MySQL	2	5000
7	PostgreSQL	1	1000
8	PostgreSQL	1	3000
9	PostgreSQL	1	5000
10	PostgreSQL	2	1000
11	PostgreSQL	2	3000
12	PostgreSQL	2	5000
13	SQL Server	1	1000
14	SQL Server	1	3000
15	SQL Server	1	5000
16	SQL Server	2	1000
17	SQL Server	2	3000
18	SQL Server	2	5000

Uma outra avaliação de desempenho foi realizada para a análise do nível de falha na requisição, com o objetivo de identificar qual é o banco de dados mais eficaz, ou seja, qual deles possui o menor nível de falha. Para esse experimento foram utilizados os mesmos SGBDs do DoE para a métrica tempo de execução: MySQL, PostgreSQL e SQL Server. Porém, o número de tuplas e número de tabelas foram fixados em 5000 tuplas e uma única tabela, pois de acordo com sucessivos testes com níveis de fatores mínimos realizados para essa avaliação, foi constatado que a alteração desses fatores não influenciam na métrica em questão.

Na métrica tempo de execução é analisada a média do tempo total (em milissegundos) de duração do experimento. Para a análise do nível de falha na requisição é observada a média da porcentagem de erro encontrada durante o experimento, ou seja, a porcentagem de solicitações que não foram atendidas. Os valores para ambas as métricas foram coletados de acordo com os dados disponibilizado pela ferramenta de teste de carga utilizada, o JMeter [1]. Escolhemos essa ferramenta, pois a mesma foi utilizada em alguns trabalhos relacionados a este, como (GILLAN et al., 2018) e (GAUR; JOSHI; SRIVASTAVA, 2017).

Para o cálculo do tempo de execução, configurou-se o Computador B com o JMeter, para que a mesma enviasse requisições de leitura para o SGBD presente no Docker,

[1] https://jmeter.apache.org/

instalado no Computador A. Para tanto, o JMeter foi configurado para gerar requisições simultâneas de leitura emulando um grupo de 100 usuários. Já na avaliação da métrica de nível de falha na requisição, o grupo de usuários sofreu variações e os teste foram realizado com 500, 1000, 1500, 2000 e 2500 threads simultâneas. Em ambas as avaliações, foram realizadas 100 réplicas de experimento para cada combinação.

4.2.2 Resultados Obtidos

Esta seção apresenta os resultados obtidos de acordo com os dados coletados no ambiente de testes. A Figura 13 apresenta o gráfico de Pareto para os fatores relacionados a métrica tempo de execução. Quando um fator possui alto impacto nos testes, significa que ao alterar o nível dele são obtidos valores bem distintos. De acordo com os valores do p-value encontrados, os efeitos do fator SGBD possui maior relevância dentre os fatores deste estudo, portanto a escolha do SGBD a ser utilizado é determinante na eficiência do contêiner, com relação ao tempo de execução.

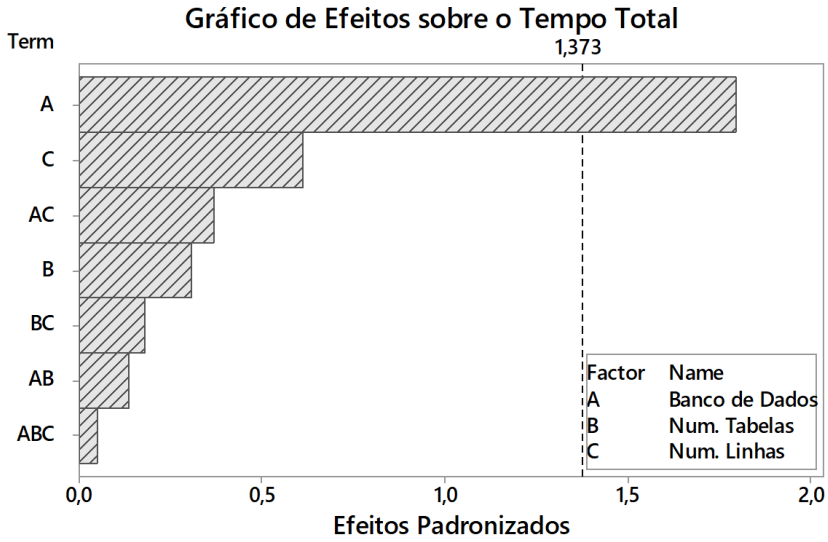

Figura 13: Influência dos fatores na métrica tempo de execução

A Figura 14 apresenta o gráfico de efeitos principais para a métrica tempo de execução. O gráfico representa o tempo médio para a realização dos testes de cada nível para cada um dos fatores definidos. Nesse gráfico, quanto mais horizontal for a linha, menos influência tem aquele fator, pois significa que os diferentes níveis do fator influenciam no resultado final de forma semelhante.

Nesse estudo, através do gráfico de efeitos principais, pode-se concluir que todos os níveis dos fatores definidos interferem na métrica tempo de execução. O SGBD e o número de tuplas são os fatores que possuem maior efeito. Sobre o fator SGBD, o MySQL atingiu a melhor média de tempo de execução (1,993ms), com o PostgreSQL bem próximo (2,031ms), enquanto o SQL Server alcançou uma média de tempo muito maior que ambos

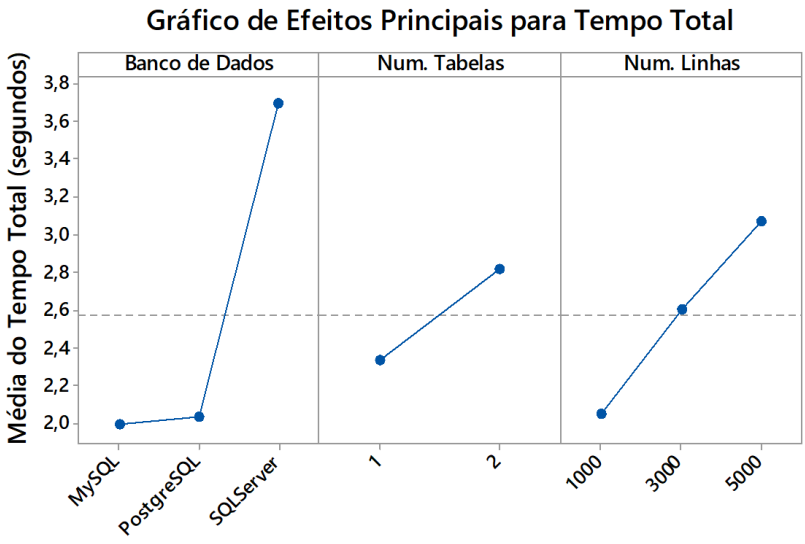

Figura 14: Efeitos principais para Tempo de Execução

(3,692ms). Sendo assim, pode-se dizer que o SGBD mais eficiente foi o MySQL, visto que alcançou a menor média de tempo de execução de leitura.

Com relação ao fator número de tabelas, percebe-se que o fator influencia no tempo médio total de acordo com a sua quantidade. Ao utilizar duas tabelas, o número de dados manipulados é dobrado, o que aumenta o tempo utilizado durante os testes. Por fim, o fator número de tuplas influencia de forma que à medida que a quantidade de tuplas aumenta, o tempo médio de leitura também aumenta. Quanto maior o número de tuplas, mais dados são manipulados, tornando necessário um tempo maior para a realização dos experimentos.

A Figura 15 exibe o gráfico de interações para cada combinação possível entre os fatores. Com esse gráfico, é possível analisar quais níveis possuem maior interferência no resultado final dos experimentos. Na primeira interação nota-se que na relação SGBD e tempo de execução, o número de tabelas influenciou no resultado. Percebe-se que ao manipular duas tabelas é obtido um tempo médio maior que o tempo de manipulação para apenas uma tabela, independente do SGBD. Da mesma forma, utilizando apenas uma tabela, apesar do desempenho ser semelhante ao caso anterior, o tempo médio total é inferior. Na segunda interação, percebe-se que o Número de Tuplas interfere no resultado da relação entre o SGBD e o Tempo de Execução. É possível constatar que o Tempo de Execução aumenta de acordo com o Número de Tuplas, independente do SGBD utilizado. No entanto, a influência do Número de Tuplas que o SQL Server sofreu foi inferior aos demais. Na última interação é analisada a relação entre o Número de Tabelas e o Número de Tuplas. Conclui-se que a influência do Número de Tabelas no Tempo de Execução cresce de acordo com o aumento do Número de Tuplas. Tal resultado acontece pois quanto maior o Número de Tuplas nas tabelas, mais dados serão manipulados, o que exige maior

processamento e consequentemente maior quantidade de tempo.

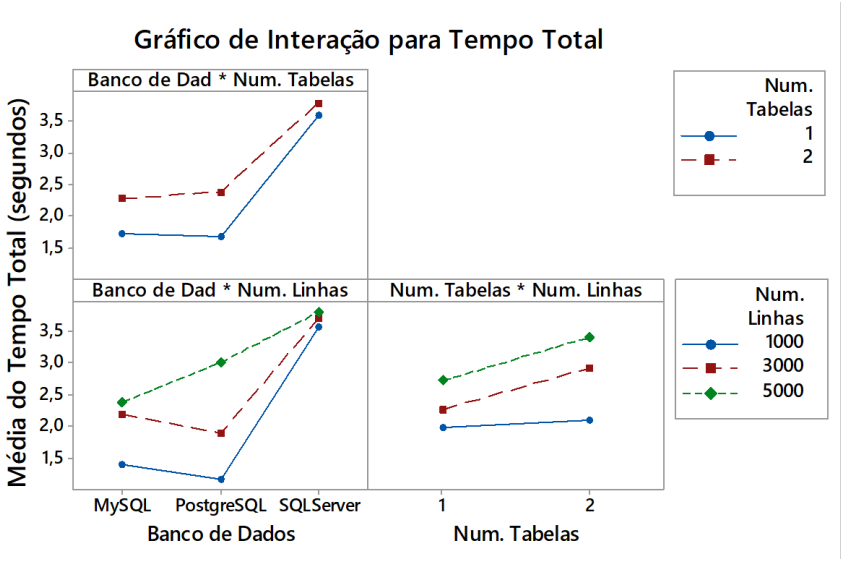

Figura 15: Interações dos fatores para Tempo de Execução

A Figura 16 apresenta o gráfico de otimização para a métrica tempo de execução. Esse gráfico mostra o efeito de cada fator na resposta do experimento. Os dados na terceira linha, que são exibidos no topo do gráfico e as linhas verticais representam as configurações atuais do fator, ou seja, os níveis que estão sendo analisados. O valor de 'y' e as linhas na horizontal representam o resultado para o nível do fator atual. Analisando o gráfico é perceptível que para obter o menor tempo possível (1,150ms), a melhor configuração para os fatores é o Banco de Dados do SGBD PostgreSQL, utilizando uma única tabela de 1000 tuplas.

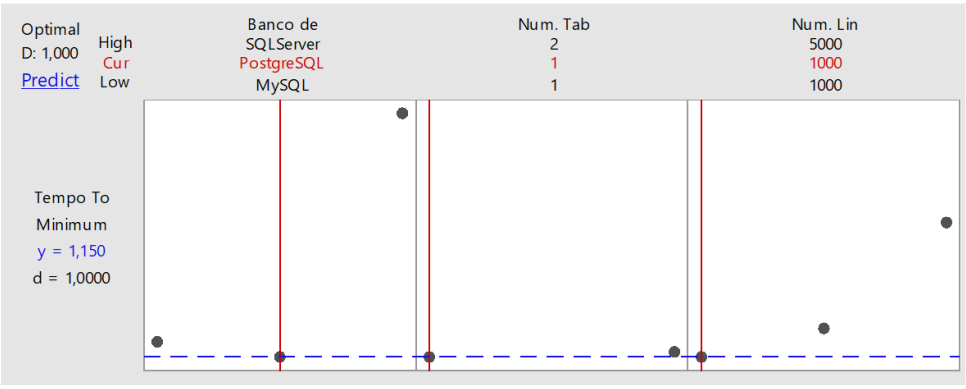

Figura 16: Gráfico de otimização para tempo de execução

A Figura 17 apresenta o gráfico de barras que exibe a porcentagem do nível de falha na requisição de cada SGBD, de acordo com a quantidade de threads simultâneas. Analisando o gráfico, percebe-se que o SQL Server obteve a menor porcentagem de falha. Já o MySQL e o PostgreSQL obtiveram porcentagem de falha semelhantes entre eles, porém muito

superior ao SQL Server. Acreditamos que a justificativa para esse resultado seja o fato de que por configuração padrão, o número máximo de conexões do MySQL e PostgreSQL é de 100 usuários simultâneos, enquanto o SQL Server esse número é dinâmico, podendo chegar a 32.767.

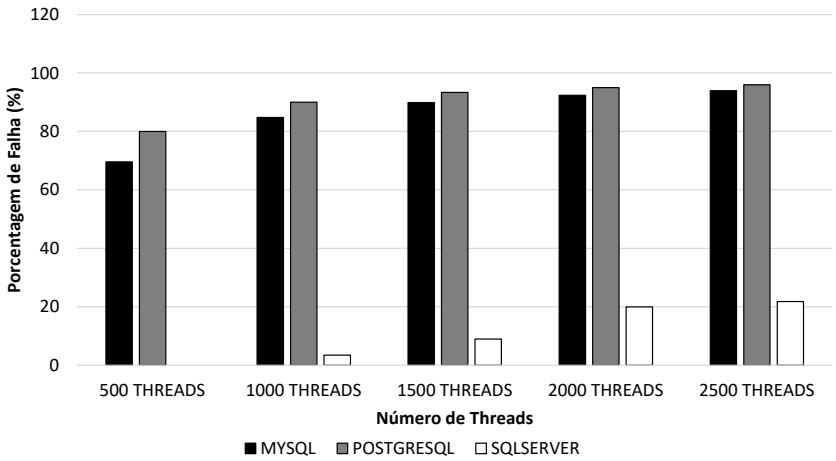

Figura 17: Porcentagem do nível de falha na requisição

A Figura 18 exibe o gráfico de barras que representa o tempo total dos testes para nível de falha na requisição, os valores para essa métrica foram coletados considerando o tempo médio das requisições bem sucedidas. Os SGBDs MySQL e PostgreSQL alcançaram os menores tempos de testes, e assim como na porcentagem do nível de falha na requisição, os resultados foram próximos um do outro. Nota-se também que o SQL Server utilizou uma quantidade de tempo muito superior aos demais, mas após observar as Figuras 17 e 18, percebe-se que apesar disso foi o SGBD que proporcionou menor número de falhas. O MySQL e o PostgreSQL mesmo com o bom desempenho relacionado ao tempo total dos testes, apresentaram grande porcentagem de falhas.

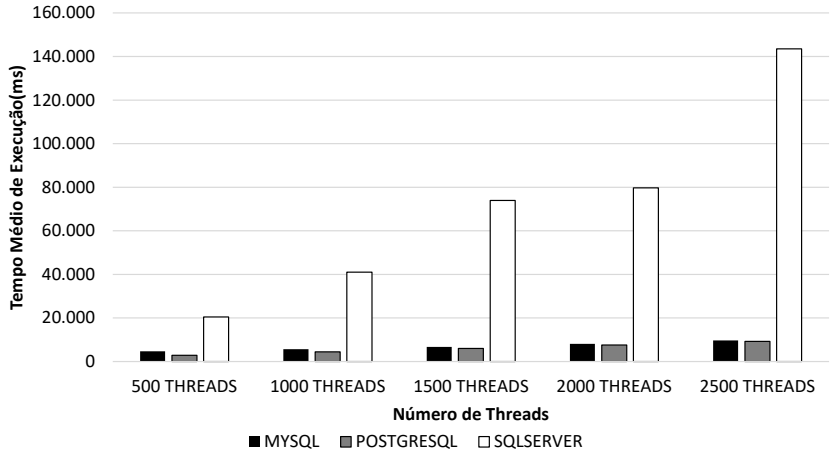

Figura 18: Tempo dos testes para nível de falha na requisição

A Figura 19 mostra as combinações que obtiveram melhor desempenho para as métricas analisadas: tempo de execução e nível de falha na requisição. As combinações foram feitas com os melhores níveis de cada fator, de acordo com as métricas utilizadas.

Figura 19: Melhores combinações obtidas

Sabendo quais são as melhores combinações, foram definidos dois cenários com a finalidade de identificar qual combinação obtém melhor desempenho, de acordo com as necessidades de cada um:

- **Cenário 1:** uma aplicação de pequeno porte precisa acessar o Banco de Dados para selecionar usuários cadastrados. Nesse caso, para obter o melhor desempenho é importante que o tempo utilizado seja o menor possível. Sendo assim, é recomendado o PostgreSQL, pois é mais eficiente com pequeno Número de Tuplas na tabela.

- **Cenário 2:** uma aplicação de porte maior e com um número de usuários superior que o cenário 1 precisa acessar o Banco de Dados para selecionar usuários cadastrados. Nesse caso, o mais importante é garantir a consistência dos dados, evitando falha na requisição dos mesmos durante a consulta. Assim, é recomendada a utilização do SQL Server, pois independente de levar mais tempo para realizar a tarefa, é o SGBD que possui menor nível de falha na requisição.

4.3 Resumo do Capítulo

As máquinas virtuais trouxeram benefícios como a possibilidade de prover tolerância a falha e balanceamento de carga. Porém, apesar das vantagens, as VMs causam perda de eficiência. Como alternativa as VMs, os contêineres se apresentam como uma solução mais leve. Este trabalho realizou uma avaliação de desempenho baseada em análise de sensibilidade com o objetivo de identificar os fatores que mais impactam na eficiência de um SGBD utilizando Docker como contêiner. Nas avaliações executadas foram observados alguns fatores, são eles: SGBD, número de tabelas e número de tuplas. De acordo com o que foi analisado, nota-se que as configurações que proporcionam melhor desempenho é relativo à métrica escolhida. Quando se trata de menor tempo de execução, a melhor combinação encontrada dentre os fatores e níveis analisados foi o uso do PostgreSQL utilizando uma única tabela com 1000 tuplas, e a pior combinação foi o uso do SQL Server com 2 tabelas e 5000 tuplas. Já para menor nível de falha na requisição, a configuração que possibilitou melhor desempenho foi o SQL Server com 500 threads simultâneas, e a pior combinação foi o PostgreSQL com 2500 threads simultâneas.

5 Conclusão

Este trabalho propôs uma avaliação de desempenho do contêiner docker, para isso, foram desenvolvidos um modelo SPN para representar e avaliar o desempenho de uma arquitetura MEC básica, e um DoE para avaliar as melhores combinações de fatores para um banco de dados. O modelo permite estimar o Tempo Médio de Resposta (MRT) e o nível de utilização dos recursos na borda da rede. O avaliador pode configurar até 5 parâmetros de entrada no modelo, o que permite um alto nível de flexibilidade de avaliação. O DoE permite identificar a melhor configuração para consultas em um banco de dados instalado no contêiner docker. Para isso, foram realizadas avaliações de desempenho para as métricas: tempo de execução e nível de falha na requisição. Nessas avaliações foram observados alguns fatores, são eles: SGBD, número de tabelas e número de tuplas.

Na primeira avaliação foi executada uma análise numérica baseando-se em dados reais de um artigo de referência que executou experimentos simplificados. A análise numérica permitiu observar o comportamento das quatro métricas (MRT, descarte, utilização master, e utilização slave) em função da variação do tempo entre chegadas de novas requisições. A análise numérica permitiu concluir que a taxa entre chegadas é um parâmetro de grande impacto sobre o sistema. Para valores muito pequenos (até 3.0ms) observou-se níveis de descarte para servidor slaves com 16 e 32 núcleos, o que pode comprometer principalmente o tempo médio de resposta. Embora tenhamos nos concentrado em um caso de uso de jogo eletrônico, os resultados de nossa análise numérica oferece *insights* sobre o uso da computação de borda para aplicativos diversos em IoT. Claro que para conclusões mais acuradas sobre outras aplicações deve-se alimentar o modelo com as mensurações respectivas da aplicação em questão.

De acordo com o que foi analisado na segunda avaliação, nota-se que as configurações que proporcionam melhor desempenho é relativo à métrica escolhida. Quando se trata de menor tempo de execução, a melhor combinação encontrada dentre os fatores e níveis analisados foi o uso do PostgreSQL utilizando uma única tabela com 1000 tuplas, obtendo o tempo total de 1,150ms, e a pior combinação foi o uso do SQL Server com 2 tabelas e 5000 tuplas, com o tempo de 3,962ms. Já para menor nível de falha na requisição, a configuração que possibilitou melhor desempenho foi o SQL Server com 500 threads simultâneas, pois proporcionou falha de 0% nas requisições, e a pior combinação foi o PostgreSQL com 2500 threads simultâneas, pois resultou em 96% de falha.

Como trabalhos futuros, pretende-se estender o modelo para incluir requisitos de disponibilidade, mensurar gasto energético e explorar alocação entre múltiplas torres MEC. Pretende-se ainda analisar outros SGBDs, bem como utilizar outros fatores e níveis para analisar mais profundamente o impacto dos mesmos na disponibilidade de um Banco de Dados.

6 Publicações

Brena Santos, Patricia Takako Endo, Francisco Airton Silva. Uma Avaliação de Desempenho de Contêineres Docker Executando Diferentes SGBDs Relacionais. **Workshop em Desempenho de Sistemas Computacionais e de Comunicação (WPerformance_CSBC)**. Acontecerá em Julho de 2019.

Brena Santos, Daniel Carvalho, Iure Fé, Francisco Airton Silva. Avaliação de Desempenho de Computação Móvel na Borda Usando Redes de Petri Estocásticas. **Workshop em Desempenho de Sistemas Computacionais e de Comunicação (WPerformance_CSBC)**. Acontecerá em Julho de 2019.

João Rosal, Rodrigo Leal, Brena Santos, Ricardo Silva, Francisco Airton Silva. Uma Plataforma Distribuída para Avaliação de Desempenho em Aplicativos Móveis. **Encontro Unificado de Computação**, Parnaíba. Piauí. 2017.

Referências

AHMED, E.; REHMANIB, M. H. Mobile edge computing: Opportunities, solutions, and challenges. *ScienceDirect*, ScienceDirect, v. 70, p. 59–63, 2017. Citado na página 13.

AMENTO, B. et al. *Systems and Methods For Allocating and Managing Resources in an Internet of Things Environment Using Location Based Focus of Attention*. [S.l.]: Google Patents, 2018. US Patent App. 15/432,042. Citado na página 11.

ANTONY, J. Taguchi or classical design of experiments: a perspective from a practitioner. *Sensor Review*, Emerald Group Publishing Limited, v. 26, n. 3, p. 227–230, 2006. Citado na página 15.

BADRI, H. et al. Multi-stage stochastic programming for service placement in edge computing systems: poster. In: ACM. *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. [S.l.], 2017. p. 28. Citado na página 20.

BARIK, R. K. et al. Performance analysis of virtual machines and containers in cloud computing. In: IEEE. *Computing, Communication and Automation (ICCCA), 2016 International Conference on*. [S.l.], 2016. p. 1204–1210. Citado 2 vezes nas páginas 29 e 30.

BECK, M. T. et al. Mobile edge computing: A taxonomy. In: CITESEER. *Proc. of the Sixth International Conference on Advances in Future Internet*. [S.l.], 2014. p. 48–55. Citado na página 18.

CAU, E. et al. Efficient exploitation of mobile edge computing for virtualized 5g in epc architectures. In: IEEE. *2016 4th IEEE international conference on mobile cloud computing, services, and engineering (MobileCloud)*. [S.l.], 2016. p. 100–109. Citado na página 18.

CHEN, X. et al. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Transactions on Networking*, IEEE, v. 24, n. 5, p. 2795–2808, 2016. Citado na página 19.

DUA, R.; RAJA, A. R.; KAKADIA, D. Virtualization vs containerization to support paas. In: IEEE. *Cloud Engineering (IC2E), 2014 IEEE International Conference on*. [S.l.], 2014. p. 610–614. Citado na página 11.

ESPEJO, M. R. *Design of experiments for engineers and scientists*. [S.l.]: Taylor & Francis, 2006. Citado na página 15.

FINK, J. Docker: a software as a service, operating system-level virtualization framework. *Code4Lib Journal*, v. 25, p. 29, 2014. Citado na página 11.

GAUR, N.; JOSHI, P.; SRIVASTAVA, R. Modelling database server sizing for concurrent users using coloured petri-nets. In: IEEE. *2017 2nd International Conference on Communication Systems, Computing and IT Applications (CSCITA)*. [S.l.], 2017. p. 90–94. Citado na página 32.

GILLAN, C. J. et al. Expediting assessments of database performance for streams of respiratory parameters. *Computers in biology and medicine*, Elsevier, v. 100, p. 186–195, 2018. Citado na página 32.

GUNST, R. F. *Response surface methodology: process and product optimization using designed experiments*. [S.l.]: Taylor & Francis, 1996. Citado na página 15.

GUPTA, P. et al. Iot based smart healthcare kit. In: IEEE. *2016 International Conference on Computational Techniques in Information and Communication Technologies (ICCTICT)*. [S.l.], 2016. p. 237–242. Citado 2 vezes nas páginas 11 e 29.

HSIEH, Y.-C. et al. Managed edge computing on internet-of-things devices for smart city applications. In: IEEE. *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*. [S.l.], 2018. p. 1–2. Citado na página 11.

HU, Y. C. et al. Mobile edge computing a key technology towards 5g. ISBH, v. 1, 2015. Citado na página 13.

JAIN, R. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. [S.l.]: John Wiley & Sons, 1990. Citado na página 25.

JARARWEH, Y. et al. The future of mobile cloud computing: integrating cloudlets and mobile edge computing. In: IEEE. *2016 23rd International conference on telecommunications (ICT)*. [S.l.], 2016. p. 1–5. Citado na página 19.

JARARWEH, Y. et al. Sdmec: Software defined system for mobile edge computing. In: IEEE. *2016 IEEE International Conference on Cloud Engineering Workshop (IC2EW)*. [S.l.], 2016. p. 88–93. Citado na página 18.

JIANG, L. et al. An iot-oriented data storage framework in cloud computing platform. *IEEE Transactions on Industrial Informatics*, IEEE, v. 10, n. 2, p. 1443–1451, 2014. Citado 2 vezes nas páginas 11 e 29.

JUNG, H. *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2015–2020 White Paper*. [S.l.], 2016. Disponível em: <http://www.cisco.com/c/en/us-/solutions/collateral/service-provider/visual-networking-indexvni/mobile-white-paper-c11-520862.html>. Citado na página 18.

KE, M. Y. Z.; S, Z. Q. L.; L, P. X. L. Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks. v. 4, p. 5896–5907, 2016. Citado na página 19.

KITANOV, S.; MONTEIRO, E.; JANEVSKI, T. 5g and the fog—survey of related technologies and research directions. In: IEEE. *2016 18th Mediterranean Electrotechnical Conference (MELECON)*. [S.l.], 2016. p. 1–6. Citado na página 18.

KUHN, D. R.; REILLY, M. J. An investigation of the applicability of design of experiments to software testing. In: IEEE. *Software Engineering Workshop, 2002. Proceedings. 27th Annual NASA Goddard/IEEE*. [S.l.], 2002. p. 91–95. Citado na página 15.

LAZIC, Z. R. *Design of experiments in chemical engineering: a practical guide*. [S.l.]: John Wiley & Sons, 2006. Citado na página 15.

LI, J. W. G.; WU, J. S. J. Data processing delay optimization in mobile edge computing. v. 2018, 2018. Citado 2 vezes nas páginas 19 e 20.

LITTLE, J. D. A proof for the queuing formula: L= λ w. *Operations research*, INFORMS, v. 9, n. 3, p. 383–387, 1961. Citado na página 23.

LIU, J. et al. Delay-optimal computation task scheduling for mobile-edge computing systems. In: IEEE. *2016 IEEE International Symposium on Information Theory (ISIT)*. [S.l.], 2016. p. 1451–1455. Citado na página 19.

MAO, Y.; ZHANG, J.; LETAIEF, K. B. Dynamic computation offloading for mobile-edge computing with energy harvesting devices. *IEEE Journal on Selected Areas in Communications*, IEEE, v. 34, n. 12, p. 3590–3605, 2016. Citado na página 19.

MARSAN, M. A. et al. *Modelling with generalized stochastic Petri nets*. [S.l.]: John Wiley & Sons, Inc., 1994. Citado na página 14.

MILLER, C. et al. Bim-extracted energyplus model calibration for retrofit analysis of a historically listed building in switzerland. *Proceedings of SimBuild*, v. 2014, 2014. Citado na página 15.

MOLLOY, M. K. On the integration of delay and throughput measures in distributed processing models. ,, University of California, Los Angeles, 1981. Citado na página 14.

MORABITO, R. A performance evaluation of container technologies on internet of things devices. In: IEEE. *Computer Communications Workshops (INFOCOM WKSHPS), 2016 IEEE Conference on*. [S.l.], 2016. p. 999–1000. Citado 2 vezes nas páginas 29 e 30.

MORABITO, R. Virtualization on internet of things edge devices with container technologies: a performance evaluation. *IEEE Access*, IEEE, v. 5, p. 8835–8850, 2017. Citado na página 11.

MURATA, T. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, IEEE, v. 77, n. 4, p. 541–580, 1989. Citado na página 13.

NIU, M. et al. Poster: Docker-based self-organizing iot services architecture for smarthome. In: ACM. *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*. [S.l.], 2017. p. 153–153. Citado na página 11.

ORSINI, G.; BADE, D.; LAMERSDORF, W. Computing at the mobile edge: Designing elastic android applications for computation offloading. In: IEEE. *2015 8th IFIP Wireless and Mobile Networking Conference (WMNC)*. [S.l.], 2015. p. 112–119. Citado na página 18.

PREMSANKAR, M. d. F. G.; TALEB, T. Edge computing for the internet of things: A case study. v. 5, p. 1275–1284, 2018. Citado 3 vezes nas páginas 19, 20 e 24.

SELTMAN, H. J. Experimental design and analysis. *Online at: http://www. stat. cmu. edu/, hseltman/309/Book/Book. pdf*, 2012. Citado na página 15.

SHIRINBAB, S.; LUNDBERG, L.; CASALICCHIO, E. Performance evaluation of container and virtual machine running cassandra workload. In: IEEE. *Cloud Computing Technologies and Applications (CloudTech), 2017 3rd International Conference of*. [S.l.], 2017. p. 1–8. Citado na página 30.

SIEBERTZ, K.; BEBBER, D. V.; HOCHKIRCHEN, T. *Statistische Versuchsplanung: design of experiments (DoE)*. [S.l.]: Springer-Verlag, 2017. Citado na página 15.

SILVA, A. C. d. Virtualização: um estudo sobre conceitos e técnicas. Universidade Federal Fluiminense, 2017. Citado na página 11.

SILVA, B. et al. Mercury: An integrated environment for performance and dependability evaluation of general systems. In: *Proceedings of Industrial Track at 45th Dependable Systems and Networks Conference (DSN)*. [S.l.: s.n.], 2015. Citado na página 14.

THOMAS, D. et al. Multiscale co-simulation of energyplus and citysim models derived from a building information model. In: *Bausim 2014: Fifth German-Austrian IBPSA Conference*. [S.l.: s.n.], 2014. Citado na página 15.

TONG, L.; LI, Y.; GAO, W. A hierarchical edge cloud architecture for mobile computing. In: IEEE. *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. [S.l.], 2016. p. 1–9. Citado na página 19.

TRINH, C.; YAO, L. Energy-aware mobile edge computing for low-latency visual data processing. p. 128–133, 2017. Citado 3 vezes nas páginas 13, 19 e 20.

VELÁSQUEZ, W.; MUNOZ-ARCENTALES, A.; RODRIGUEZ, J. S. A case study: Ingestion analysis of wsn data in databases using docker. In: IEEE. *2018 1st International Conference on Computer Applications & Information Security (ICCAIS)*. [S.l.], 2018. p. 1–6. Citado 3 vezes nas páginas 11, 29 e 30.

WANG, S. et al. A survey on mobile edge networks: Convergence of computing, caching and communications. *IEEE Communications Surveys and Tutorials*, IEEE, v. 5, p. 6757–6779, 2017. Citado na página 13.

XAVIER, M. G. et al. Containers or hypervisors, which is better for database consolidation? 2016. Citado na página 11.

YUANZHE; SHANGGUANG. An energy-aware edge server placement algorithm in mobile edge computing. In: *IEEE International Conference on Edge Computing (EDGE)*. [S.l.: s.n.], 2018. Citado na página 20.

TERMO DE AUTORIZAÇÃO PARA PUBLICAÇÃO DIGITAL NA BIBLIOTECA "JOSÉ ALBANO DE MACEDO"

Identificação do Tipo de Documento

( ) Tese

( ) Dissertação

(X) Monografia

( ) Artigo

Eu, Brena Maia Santos,

autorizo com base na Lei Federal nº 9.610 de 19 de Fevereiro de 1998 e na Lei nº 10.973 de 02 de dezembro de 2004, a biblioteca da Universidade Federal do Piauí a divulgar, gratuitamente, sem ressarcimento de direitos autorais, o texto integral da publicação Avaliação de Desempenho de Contêineres sob Dois Aspectos: Planejamento de Arquiteturas e Armazenamento de Dados.

de minha autoria, em formato PDF, para fins de leitura e/ou impressão, pela internet a título de divulgação da produção científica gerada pela Universidade.

Picos-PI 20 de junho de 2019.

Brena Maia Santos
Assinatura

Brena Maia Santos
Assinatura