Detecção de Overbooking em Aplicações Baseadas em Docker Através de Aprendizagem de Máquina

Pedro Horchulhack[1], **Eduardo Viegas**[1], **Altair Santin**[1], **Felipe Ramos**[1]

[1]Programa de Pós-Graduação em Informática (PPGIa)
Pontifícia Universidade Católica do Paraná (PUCPR)
Curitiba – PR

{pedro.horchulhack,eduardo.viegas,santin,felipe.ramos}@ppgia.pucpr.br

Resumo. *O artigo propõe um modelo de aprendizado de máquina para detectar ambientes Kubernetes com overbook de recursos em um contêiner do Docker. As métricas do aplicativo e do sistema foram coletadas continuamente, as quais foram usadas como entrada para o modelo para identificar interferência causada por multi-tenancy. Os experimentos foram executados em um cluster Kubernetes, com um aplicativo de Big Data baseado em contêiner, o que mostrou que o modelo pode detectar overbooking de recursos com precisões de até* 98% *em um cenário que pode causar degradação no desempenho do aplicativo, com taxas de overbooking de até* $1,2$.

1. Introdução

Para aumentar o lucro e otimizar a alocação de recursos, provedores públicos de nuvem computacional (NC) frequentemente alocam mais recursos virtuais do que recursos físicos disponíveis. Essa situação, conhecida como *overbooking*, leva em consideração o fato de recursos virtuais alocados requisitados por *tenants* na NC estarão a maior parte do tempo em estado ocioso. Portanto, os recursos físicos podem ser providos para outros *tenants* concorrentemente.

Ovebooking de recursos pode causar degradação na Qualidade de Serviço *(Quality of Service, QoS)* para *tenants* na nuvem caso outros clientes requisitem seus recursos virtuais concorrentemente, caso não haja recursos físicos o suficiente para lidar com as necessidades de processamento [Hoeflin and Reeser 2012]. Os *hypervisors* frequentemente implementam algoritmos de justiça para lidar com *overbooking*, enquanto nas VMs em estado ocioso receberão maior prioridade quando uma carga elevada de processamento é requisitada. No entanto, *overbooking* em um cenário de orquestração de contêineres é um problema difícil, porque o problema de múltiplos-inquilinos é gerenciado pelo SO hospedeiro, onde ele trata os contêineres como processos tradicionais do espaço do usuário [Zhong and Buyya 2020]. Apesar de ser um desafio bastante conhecido e bastante estudado em relação as NC baseadas em VMs, *overbooking* de recursos em serviços de contêineres ainda permanece esquecido na literatura [Truyen et al. 2016].

O presente artigo visa propor um modelo de aprendizagem de máquina para identificar *overbooking* de recursos de dentro do domínio do cliente em um ambiente de orquestração de contêineres Kubernetes. O modelo proposto é implementado em duas etapas. Na primeira, é monitorado periodicamente e continuamente o contêiner coletando métricas de aplicação e métricas de uso SO *contêinerizadas*. Nossa abordagem assume que conseguimos detectar problemas relacionados a *overbooking* analisando a

performance da aplicação e o uso de recursos do SO. Na segunda, nós tratamos *overbooking* de recursos como um problema de classificação através de técnicas de aprendizagem de máquina.

2. Trabalhos relacionados

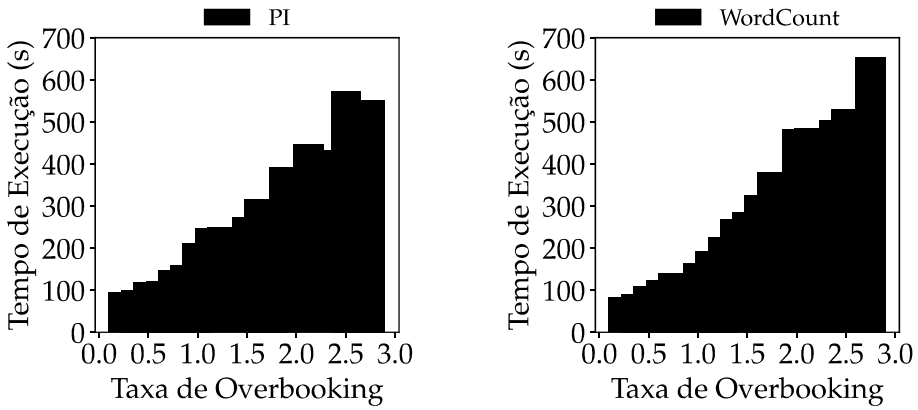
Overbooking de recursos é uma abordagem bastante conhecida e utilizada em ambientes de computação em nuvem para melhorarem o uso de hardware físico [Duc et al. 2019]. Geralmente, pesquisas são realizadas para identificarem abordagens mais eficientes para overbooking de recursos sem que haja perna na qualidade de serviço, portanto, na perspectiva do provedor de nuvem computacional [Abreu et al. 2020]. Por exemplo, Caglar e Gokhale [Caglar and Gokhale 2014] propõe uma técnica utilizando um modelo de aprendizagem de máquina para prever o uso de recurso de um inquilino na nuvem para um melhor overbooking de recursos. Os autores foram capazes de aumentar o uso de recursos físicos em nuvens IaaS, sem causar impacto na qualidade de serviço. Por outro lado, Hoeflin e Reeser [Hoeflin and Reeser 2012] propõe um modelo analítico para melhorar o uso de recursos em hardware físico em IaaS. Os autores foram capazes de preservar o consumo de energia elétrica do provedor da NC. Tordsson [Tomás and Tordsson 2014] propõe uma nova configuração de NC para habilitar o mapeamento de serviços críticos de um cliente, referente ao aperfeiçoamento da performance de uso de núcleos de CPU físico. Idealmente, a identificação de overbooking de recursos deve ser feita pelo lado do cliente devido ao conflito de interesses. Por exemplo, uma técnica de aprendizagem de máquina para a identificação de overbooking foi proposta por C. Vicentini *et al.* [Vicentini et al. 2018]. A abordagem dos autores realiza benchmarks periodicamente dentro de um cliente em uma máquina virtual para identificar desvios na performance como uma técnica de classificação. Venkateswaran e Sarkar [Venkateswaran and Sarkar 2019] propõe um SLA de posicionamento de VM *bare-metal* para fornecer garantias de performance. Por outra perspectiva, Truyen *et. al.* [Truyen et al. 2016] propõe um novo modelo de SLA para serviços em contêineres, que também não leva em consideração o overbooking de recursos.

3. Definição do Problema

A presente seção procura investigar o impacto de múltiplos inquilinos e overbooking de recursos em aplicações *dockerizadas*. Para isso, instanciamos uma aplicação de processamento de Big Data em um ambiente contêinerizado e distribuído através do Kubernetes. Os próximos itens descrevem os testes realizados e problemas relacionados a performance devido ao overbooking.

3.1. Testes

Para avaliar a degradação na performance devido ao overbooking de recursos, nós implantamos um ambiente de orquestração de contêineres distribuído através do Kubernetes $v1.19$ e Docker $v19.03.13$. Quatro nós físicos compõem a bateria de testes, onde um nó foi utilizado como Kubernetes *master* e os demais como *workers*, como tipicamente feito na literatura [Horchulhack et al. 2022]. Os nós são equipados com um *CPU Intel i7* com 8 núcleos, $16GB$ de memória, interconectados por uma rede gigabit, no sistema operacional Ubuntu $v18.04$. Como caso de uso de aplicação, consideramos uma versão



(a) Tarefa PI **(b)** Tarefa WordCount

Figura 1. Impacto na performance das tarefas avaliadas.

contêinerizada e distribuída de um cluster de Apache Spark $v3.0.0$, composto por quatro contêineres, um *master* e os demais como *workers* [Viegas et al. 2021]. Os *workers* foram configurados para utilizar somente 2 núcleos de CPU. Cada instância da aplicação fora instanciada como um Pod no Kubernetes. Duas aplicações foram avaliadas, como descritas a seguir:

- ***Tarefa PI*** Uma tarefa distribuída do Apache Spark, *contêinerizada*, que computa o valor de PI com até 10.000 casas decimais de precisão. A principal demanda desse item é o consumo de CPU;
- ***Tarefa WordCount*** Uma tarefa distribuída do Apache Spark, *contêinerizada*, que computa as ocorrências de palavras em um arquivo de $500MB$. As demandas desse item é o consumo de CPU, disco e rede;

Para criar uma situação de overbooking de recursos, para cada execução, nós variamos a quantidade de inquilinos (contêineres) executando concorrentemente no Kubernetes. Para esse fim, antes de executarmos as tarefas do Apache Spark, instanciamos um Pod Kubernetes que cria contêineres para executar benchmarks de CPU, *single-threaded*, através da ferramenta *sysbench*.

3.2. O impacto de múltiplos inquilinos em contêineres

A Figura 1 mostra o tempo de processamento para cada tarefa do Apache Spark avaliada, de acordo com a taxa de overbooking. É notável que há degradação no tempo de processamento de ambas as tarefas avaliadas quando a taxa de $0,75$ de overbooking é atingida, com maior impacto se o provedor realizar $1,0$ de taxa de overbooking. Por exemplo em um overbooking de $2,0$, a tarefa PI aumenta seu tempo de processamento em até 75%, enquanto a tarefa WordCount pode atingir até 196% de aumento. Evidentemente, o aumento de somente $0,1$ na taxa de overbooking implica, em média, um acréscimo de 11% e 14% no tempo de processamento de cada tarefa, PI e WordCount, respectivamente.

4. Modelagem de algoritmos de AM para detecção de overbooking em aplicações baseadas em contêineres no Kubernetes

A presente seção apresenta um novo modelo de aprendizagem de máquina para a detecção de overbooking de recursos de dentro do domínio do cliente, em um ambiente de

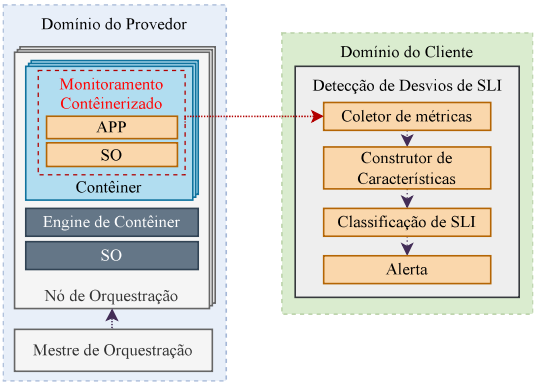


Figura 2. Modelo proposto.

orquestração de contêineres. É composto de duas etapas principais, chamadas *Monitoramento Contêinerizado* e *Detecção de Desvios de SLI*, como apresenta a Figura 2, e procura monitorar ambas métricas de aplicação e de SO, do contêiner (docker), para a identificação de overbooking de recursos.

A proposta considera um serviço de orquestração de contêiner onde um cliente deseja monitorar sua aplicação para identificar overbooking de recursos em tempo real. Dito isso, o cliente não possui acesso a infraestrutura da NC, incluindo o SO hospedeiro. O monitoramento pode ser realizado, somente, se for de dentro do domínio do cliente através do monitoramento da sua aplicação e/ou SO. Tal etapa objetiva encontrar desvios de SLI em tempo real, como por exemplo, um provedor de serviço que não fornece o tempo de CPU esperado para o contêiner do cliente. Nossa proposta considera a identificação de desvios SLI como uma tarefa de aprendizagem de máquina supervisionada. Portanto, um conjunto de métricas *contêinerizadas*, como uso de recursos do SO e performance de aplicação, é utilizado como entrada para um classificador de aprendizagem de máquina que detecta desvios de SLI.

4.1. Monitoramento *Contêinerizado*

O objetivo principal do módulo *Monitoramento Contêinerizado* é monitorar ambas as métricas de performance do SO e da aplicação continuamente. A justificativa da nossa abordagem é que ambas as métricas podem ser utilizadas para a identificação de problemas de performance de dentro do domínio do Docker. É assumido que um modelo de aprendizagem de máquina específico para classificação pode ser aplicado para detecção de desvios de SLI através da análise das métricas de utilização de recursos do SO e da aplicação. Em um cenário com interferência de múltiplos inquilinos, as métricas de uso do SO serão altas, enquanto a aplicação irá executar de maneira insuficiente.

O módulo *Monitoramento Contêinerizado* é composto por duas entidades nomeadas *Coletor APP* e *Coletor SO*, que coletam as métricas periodicamente, dentro do contêiner Docker. A primeira extrai métricas de performance de aplicação, enquanto a segunda extrai métricas de utilização de recursos do SO. Como resultado, *Monitoramento Contêinerizado* extrai dois conjuntos de características relacionadas a performance da aplicação e uso de recursos de SO, ambas de dentro do domínio do cliente.

4.2. Detecção de desvios de SLI

O processo de classificação é executado periodicamente e continuamente em um ambiente de domínio do cliente (Detecção de Desvios de SLI, Figura 2). O domínio do cliente é executado fora do domínio do provedor. Portanto, a situação não parece propensa de causar um conflito de interesses com o provedor. A classificação inicia com o módulo *Coletor de Métricas* que coleta ambas *Métricas de APP* e *Métricas de SO* do módulo de *Monitoramento Contêinerizado* (Seção 4.1) de um conjunto de contêineres sendo executados no domínio do provedor. As métricas são coletadas periodicamente a um intervalo de tempo pré-definido, por exemplo a cada 5 segundos. O conjunto de métricas coletadas de cada contêiner é enviado para o módulo *Construtor de Características* cujo objetivo é compor um vetor de características para classificação. Um conjunto de características é construído, onde cada vetor é composto por ambas *Métricas de APP* e *Métricas de SO* de um único contêiner. Posteriormente, o conjunto é direcionado ao módulo *Classificação de SLI*, que executa um classificador de aprendizagem de máquina para classificar cada vetor de características como um cenário de overbooking ou normal. Cenários classificados como overbooking são contêineres experienciando interferência de múltiplos inquilinos devido ao overbooking de recursos do provedor da nuvem computacional. Como resultado, o modelo proposto foi capaz de identificar interferência múltiplos inquilinos de dentro do ambiente contêinerizado.

5. Protótipo

O protótipo foi implementado na mesma bateria de testes avaliados anteriormente (Seção 3.1), através do Kubernetes, e uma aplicação do cliente executando um cluster do Apache Spark, composto por três *workers* e um *master*. Cada contêiner do cliente executa ambos os módulos *Coletor de Métricas de APP* e *Coletor de Métricas de SO*.

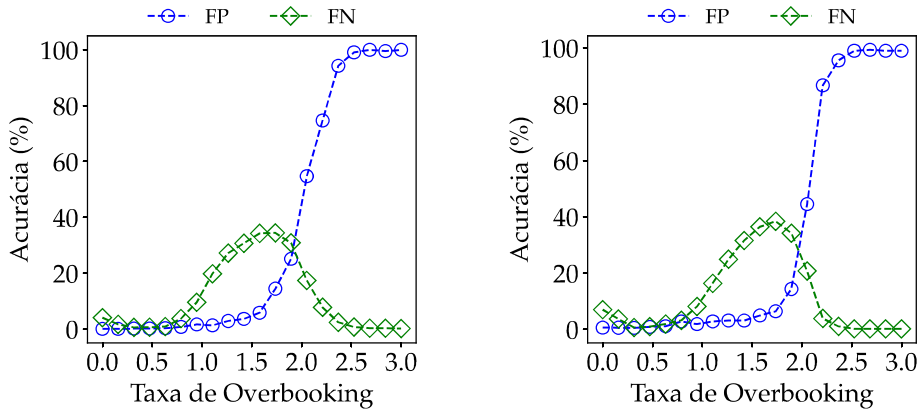
O primeiro coleta cinco características relacionadas a performance da aplicação, através da API do Apache Spark, enquanto o segundo coleta oito características de SO *contêinerizadas* através da *API PSUtil* $v5.7.2$, para a linguagem Python $v3.8$. As características são coletadas de cada contêiner em um intervalo de tempo de 5 segundos. A cada tempo de intervalo, o módulo *Coletor de Métricas* recebe as métricas coletadas através de um *web service SOAP*, implementado através da *API SOAPpy* $v0.12.22$. Um classificador de aprendizagem de máquina classifica cada vetor de características, de cada contêiner, através da *API scikit-learn* $v0.23$.

6. Avaliação

A avaliação visa responder as seguintes perguntas de pesquisa: (RQ1) *A proposta é capaz de detectar overbooking de recursos de dentro do domínio do cliente?* (RQ2) *Qual é o mínimo de overbooking de recursos necessário para uma detecção precisa?* (RQ3) *Qual o atraso na detecção para a identificação de overbooking de recursos?*

6.1. Detecção de overbooking de recursos

Para avaliar o modelo proposto, o mesmo conjunto de experimentos executados na Seção 3.1 foram feitos com o nosso protótipo implementado. Cada configuração de grau de overbooking, de $0,25$ a $3,0$, foram avaliados, onde as tarefas do Apache Spark foram monitoradas por 10 minutos de tempo de execução em cada cenário. Portanto, cada



(a) Tarefa PI **(b)** Tarefa WordCount

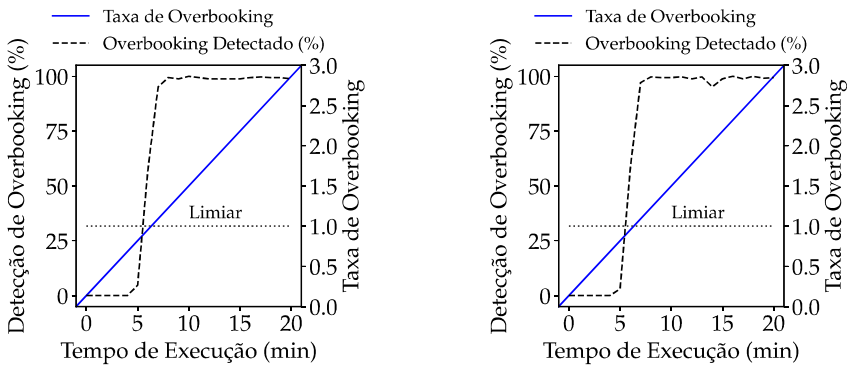
Figura 3. Acurácia do classificador GBT.

Conj. Caract.	Classificador	Acurácias (%)			
		Tarefa PI		Tarefa WordCount	
		FP	FN	FP	FN
Todas as Caract.	DT	5,55	10,99	5,26	11,16
	RF	1,58	9,56	2,48	7,99
	GBT	1,65	9,56	1,84	8,13
Sel. de Caract.	DT	5,41	11,12	4,91	10,61
	RF	1,44	10,08	2,28	8,27
	GBT	1,44	10,08	**1,92**	**8,30**

Tabela 1. Acurácias da abordagem proposta.

cada configuração de overbooking avaliada, uma média de 120 (5 por segundo) vetores de características foram coletados de cada contêiner *Worker* do Apache Spark.

O primeiro experimento procura responder a pergunta RQ1 e avaliar a acurácia do modelo proposto para identificação de overbooking de recursos do domínio do cliente. Nós valiamos quatro classificadores comuns de aprendizagem de máquina, a Árvore de Decisão (Decision Tree, DT) com um fator de confiança de $0,25$; os classificadores Random Forest (RF) e Gradient Boosting (GBT) com 100 árvores de decisão como estimadores base [Bulle et al. 2020]. Cada classificador foi avaliado com e sem seleção de características. Para isso, a seleção de características aplica uma técnica que filtra baseado no ganho de informação, onde somente características com ganhos acima de $0,2$ são usadas para o treinamento dos classificadores. Duas classes foram utilizadas, *normal* e *overbooking*. Para o processo de treinamento dos modelos, as métricas coletadas de dois *workers* (contêineres) são utilizados para esse fim, enquanto que o restante é utilizado para teste. Os classificadores foram avaliados através das suas taxas de Falso-Positivo (FP) e Falso-Negativo (FN). A taxa de FN significa que valores classificados como *normal* foram erroneamente classificados como normal, ou seja, há overbooking. Já a taxa FP demonstra a proporção dos vetores classificados erroneamente como *overbooking*, sendo eles eventos *normais*.



(a) Tarefa PI **(b)** Tarefa WordCount

Figura 4. Taxas de detecção de overbooking do modelo GBT.

A Tabela 1 mostra as acurácias de classificação da proposta, para cada classificador avaliado, para cada tarefa do Apache Spark, para a detecção de overbooking de recursos em cenários com limiares de overbooking acima de $1,0$; É possível notar que a nossa proposta foi capaz de fornecer altas acurácias na detecção, com até $1,92\%$ de FP e $8,30\%$ de FN para a tarefa WordCount com o classificador GBT, com todas as características, enquanto atingiu $1,65\%$ de FP e $9,56\%$ de FN para a tarefa PI. Para responder a pergunta RQ2, nós avaliamos a sensibilidade de classificação em overbooking de recursos. Nós variamos os rótulos de cada cenário como *normal* ou *overbooking* de acordo com a sua taxa de overbooking, assim o operador deve definir a sensibilidade de acordo com suas necessidades. A Figura 3 mostra as acurácias das propostas de acordo com a taxa de overbooking com o classificador GBT com todas as características (classificador mais preciso, Tabela 1). É possível notar que a proposta é capaz de identificar, com altas taxas de acurácia, interferência causada por overbooking que poderão causar um aumento significativo no tempo de processamento (quando a taxa de overbooking passa a marca de $1,2$, como avaliado na Seção 3).

Finalmente, para responder a questão RQ3, nós instanciamos nosso modelo proposto, com uma limiar de $1,0$ de overbooking e investigamos a sensividade na detecção de overbooking enquanto varia-se interferência por múltiplos inquilinos concorrentemente. A Figura 4 mostra a acurácia da proposta ao longo do tempo e o intervalo de detecção a cada mudança na taxa de overbooking. É evidente que nossa proposta é capaz de detectar interferência de múltiplos inquilinos nos primeiros momentos em que a tarefa do Apache Spark começa a perder performance, depois da limiar de $1,0$ de overbooking ser ultrapassada. Ademais, é possível diminuir o tempo de detecção através da diminuição do período de coleta das características (5 segundos em nosso protótipo).

7. Conclusão

Overbooking de recursos é um desafio conhecido em ambientes tradicionais de nuvem computacional, no entanto permanece esquecido em problemas relacionados a ambientes *contêinerizados*. Este artigo propôs uma nova abordagem para identificar overbooking de recursos de dentro do domínio do cliente, que implica perda na qualidade de serviço. O modelo proposto foi capaz, através de aplicação de técnicas de aprendizagem de máquina através de métricas de performance de aplicação e de SO, identificar

com altas taxas de acurácia quando o provedor de nuvem computacional está praticando o overbooking de recursos que pode afetar a performance no processamento de serviços *contêinerizados*. Como trabalhos futuros, nós iremos avaliar o modelo proposto em ambientes *contêinerizados* em cenários mais dinâmicos.

Referências

Abreu, V., Santin, A. O., Viegas, E. K., and Cogo, V. V. (2020). Identity and access management for IoT in smart grid. In *Advanced Information Networking and Applications*, pages 1215–1226. Springer International Publishing.

Bulle, B. B., Santin, A. O., Viegas, E. K., and dos Santos, R. R. (2020). A host-based intrusion detection model based on OS diversity for SCADA. In *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society*. IEEE.

Caglar, F. and Gokhale, A. (2014). ioverbook: Intelligent resource-overbooking to support soft real-time applications in the cloud. In *2014 IEEE 7th International Conference on Cloud Computing*, pages 538–545.

Duc, T. L., Leiva, R. G., Casari, P., and Östberg, P.-O. (2019). Machine learning methods for reliable resource provisioning in edge-cloud computing: A survey. *ACM Comput. Surv.*, 52(5).

Hoeflin, D. and Reeser, P. (2012). Quantifying the performance impact of overbooking virtualized resources. In *2012 IEEE International Conference on Communications (ICC)*, pages 5523–5527.

Horchulhack, P., Viegas, E. K., and Santin, A. O. (2022). Toward feasible machine learning model updates in network-based intrusion detection. *Computer Networks*, 202:108618.

Tomás, L. and Tordsson, J. (2014). Cloud service differentiation in overbooked data centers. In *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, pages 541–546.

Truyen, E., Van Landuyt, D., Reniers, V., Rafique, A., Lagaisse, B., and Joosen, W. (2016). Towards a container-based architecture for multi-tenant saas applications. In *International Workshop on Adaptive and Reflective Middleware*, ARM 2016.

Venkateswaran, S. and Sarkar, S. (2019). Time-sensitive provisioning of bare metal compute as a cloud service. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pages 447–451.

Vicentini, C., Santin, A., Viegas, E., and Abreu, V. (2018). A machine learning auditing model for detection of multi-tenancy issues within tenant domain. In *ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 543–552.

Viegas, E., Santin, A. O., and Jr, V. A. (2021). Machine learning intrusion detection in big data era: A multi-objective approach for longer model lifespans. *IEEE Transactions on Network Science and Engineering*, 8(1):366–376.

Zhong, Z. and Buyya, R. (2020). A cost-efficient container orchestration strategy in kubernetes-based cloud computing infrastructures with heterogeneous resources. *ACM Trans. Internet Technol.*, 20(2).