Arquitetura para escalonamento oportunístico na nuvem com qualidade de serviço

Caetano Albuquerque[1], **Diego Gama**[1], **Andrey Brito**[1]

[1]Laboratório de Sistemas Distribuído – Universidade Federal de Campina Grande (UFCG)
Campina Grande, PB – Brasil

{caetano.albuquerque,diegogama,andrey}@lsd.ufcg.edu.br

Abstract. *The use of opportunistic computing on the cloud is popular due to enabling hosting batch processing applications at lower costs. However, this model does not guarantee availability, which restricts quality of service. Moreover, since prices fluctuate frequently, high priority on optimal availability requires low priority on cost savings. This paper proposes a modular opportunistic auto-scaler for Kubernetes that balances opportunistic node costs and processing delay to ensure quality of service. Based on prices from 2021 AWS, the solution provided up to 50% savings when comparing to on-demand prices, thus allowing near-realtime processing.*

Resumo. *O uso de computação oportunista na nuvem é popular por permitir hospedar aplicações com processamento em lotes a custo reduzido. Todavia, esse modelo não garante disponibilidade, o que restringe a qualidade de serviço dessas aplicações. Além disso, graças à alta oscilação de preços, atingir uma disponibilidade ótima com esse tipo de serviço requer despriorizar potencial econômico. Este artigo propõe um escalonador oportunístico modular para Kubernetes que equilibra custos de nós oportunistas e atrasos no processamento para garantir qualidade de serviço. Com base em preços de 2021 da AWS, a solução proporcionou economia significativa de até 50% em relação ao custo de instâncias sob demanda, permitindo processamento em tempo quase real.*

1. Introdução

O uso de nuvens públicas tem se tornado bastante popular graças ao modelo IaaS (*infrastructure as a service*), que permite que contratantes implantem suas aplicações na nuvem para poupar esforços e custos operacionais. Contratantes podem dispor de uma grande variedade de instâncias disponíveis, nas quais podem hospedar seus serviços a depender de seus requisitos. Assim conseguem delegar as preocupações com a infraestrutura e direcionar seus esforços no produto, de maneira a fornecer QoS (*Quality of Service*, i.e. qualidade de serviço) a seus clientes.

O modelo mais tradicional para alocação de instâncias é o sob demanda, no qual um provedor oferta alta disponibilidade mediante um custo por tempo de uso (como por minuto ou até por segundo). Todavia, um modelo alternativo a esse, com grande potencial de economia, é o oportunista. Essas instâncias se destacam das mais tradicionais instâncias sob demanda, pois representam a possibilidade de provedores monetizarem recursos ociosos da nuvem sem garantir disponibilidade. Assim, o conjunto de recursos ociosos disponíveis que compõe uma instância Spot (oportunista) oscila, conforme

instâncias tradicionais são mais ou menos contratadas. Como incentivo para utilizá-las, provedores calculam seus preços como pequenas parcelas do preço de uma instância sob demanda. Isso faz com que seja possível reduzir custos de infraestrutura para aplicações que toleram as inevitáveis faltas de disponibilidade.

Uma vez que a disponibilidade de instâncias oportunistas não é garantida, contratantes são limitados pelo tipo de aplicações que podem implantar. Sistemas cujos requisitos de QoS estabelecem, por exemplo, disponibilidade mínima de acesso ou prazos para processamento de carga, não são compatíveis com oportunismo. Isso limita o potencial do uso desses recursos, que por sua vez restringe as oportunidades de economia em infraestrutura. Conforme as instâncias oportunistas se tornam mais escassas, elas também se tornam mais caras. Portanto, para maximizar a possibilidade de alocação dessas instâncias é necessário lidar com a instabilidade de seus preços.

Para reduzir as restrições no uso de computação oportunista é preciso que haja um mecanismo capaz de balancear a alocação dos recursos, de maneira a evitar que termos de QoS sejam violados. Este artigo busca tratar desse problema ao propor a arquitetura de um escalonador oportunista, que escalona aplicações em um *cluster* Kubernetes cujos nós trabalhadores são hospedados em instâncias oportunistas. Nós e aplicações são escalonados conforme necessário para garantir um equilíbrio entre atraso no processamento de carga e economia no custo da infraestrutura. Isso viabiliza o uso de oportunismo para o processamento em tempo quase real, isto é um processamento com prazo curto, próximo ao imediato, e oferece controle de gastos ao evitar que o custo por instância exceda um orçamento configurado.

Além disso, visto que por definição a nuvem se trata de um ambiente cuja responsabilidade da infraestrutura é do provedor, esse ambiente torna-se não confiável. Requisitos de segurança podem desincentivar ainda mais o uso de computação na nuvem para certos sistemas. Aplicações que lidam com dados sensíveis, por exemplo, precisam da garantia que seus dados serão protegidos. Ameaças cada vez mais comuns à cadeias de suprimento de *software* (com crescimento de até 650% cumulativos até 2021 [Okafor et al. 2022]), como o ataque à SolarWinds em 2020 [Peisert et al. 2021], enfatizam essa preocupação. Medidas de segurança não são cobertas por esse artigo, mas guias fortes como SLSA da Google e Secure Software Factory da CNCF podem ser seguidos para fortalecer a segurança de aplicações sensíveis na nuvem, e potencialmente permitir o uso de aplicações assim em um modelo oportunista. Neste artigo focamos no uso de computação oportunística para uma aplicação de análise de dados de energia.

Este artigo está organizado nas seguintes seções. Seção 2, na qual são detalhadas as problemáticas apresentadas na introdução; Seção 3, onde é detalhada uma arquitetura para orquestração oportunista; Seção 4, em que a arquitetura é validada com auxílio de uma simulação; e Seção 5, que conclui os benefícios e limitações da solução proposta.

2. Fundamentação teórica

2.1. Computação oportunista

Existem dois principais modelos de cobrança para contratação de VMs (*virtual machines*), adotado pela maioria dos provedores de nuvem: os planos com preços estáticos (instâncias reservadas e sob demanda) e os com custos dinâmicos (instâncias oportunistas)

[Kumar et al. 2017]. Em planos sob demanda, o provedor aloca uma ou mais instâncias e as mantém disponíveis pela duração do plano. Já em planos de instâncias reservadas, o contratante assume um compromisso duradouro (de ao menos um ano) em troca de um preço reduzido em comparação ao plano sob demanda. Em ambos os planos o contrante tem controle sobre a disponibilidade e o preço estático de cada instância alocada.

Além desses dois planos, provedores oferecem instâncias oportunistas para aproveitar capacidade computacional não utilizada da infraestrutura. Essas VMs costumam ser ofertadas a um preço promocional (p. ex., 90% de desconto no caso da AWS [Services 2022]), como incentivo para alocação. O uso de computação oportunista é uma prática popular para algumas atividades de processamento de lote. Como por exemplo, o uso de Hadoop aliado ao *spot market* da AWS para trabalhos de Big Data com *deadlines* [Tamrakar et al. 2017]. A maior desvantagem desse tipo de instância é que pela sua natureza oportunista, o provedor pode a qualquer momento desligar as instâncias alocadas pelo contratante caso os recursos que as compõem deixem de ser ociosos, com aviso prévio de poucos minutos.

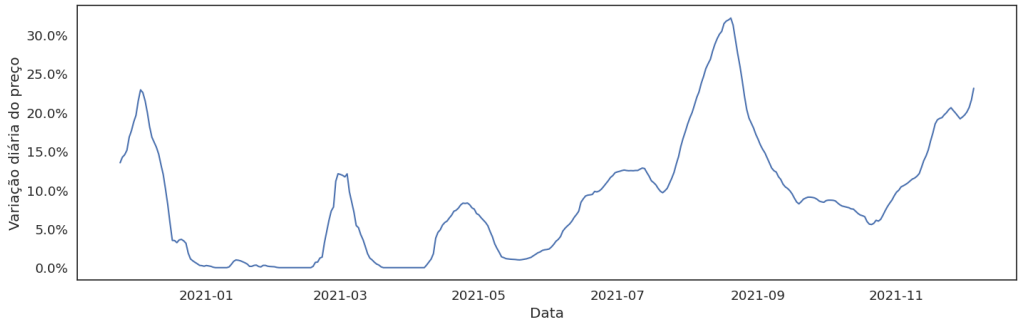

Figura 1. Variação diária no preço da instância Spot `c6.4xlarge` da AWS, normalizada pelo preço no plano sob demanda.

Uma característica importante das instâncias oportunistas é que seus preços variam de acordo com o quão ociosos estão os recursos na infraestrutura, podendo oscilar diversas vezes durante o dia. A Figura 1 acima ilustra a variação diária do preço da instância oportunista `c6g.4xlarge` da AWS em 2021. Provedores oferecem a opção de definir um custo limite a ser pago pela instância, e a desligam caso o preço alcance o orçamento, a fim de ofertar ao contratante um maior controle sobre os gastos. Porém, visto que não há garantia de quando a instância irá retornar a um preço aceitável, esse comportamento traz ainda mais imprevisibilidade acerca da disponibilidade da instância. Um possível artifício para contornar essa imprevisibilidade seria definir estaticamente um horário específico para alocar as instâncias e executar a aplicação.

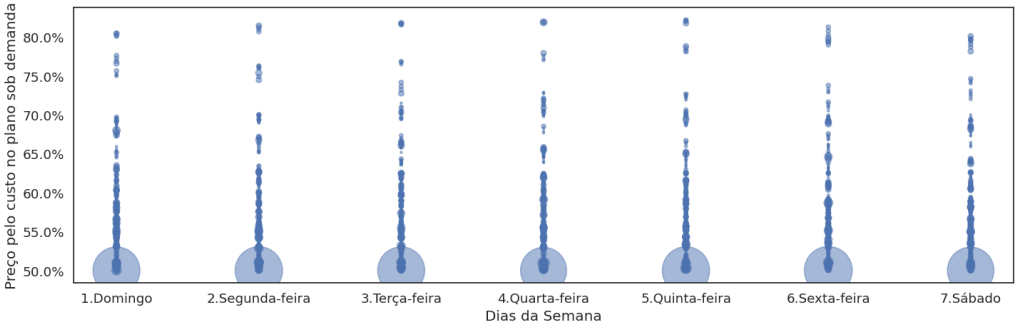

Figura 2. Preço horário da instância Spot `c6.4xlarge` da AWS, normalizado pelo preço no plano sob demanda.

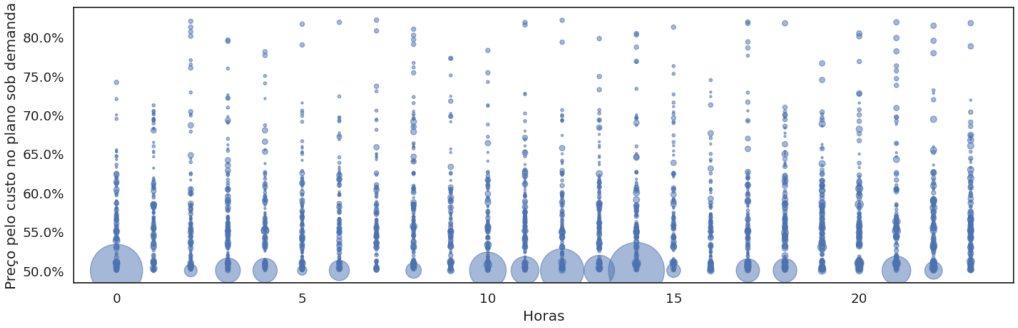

Figura 3. Preço da instância Spot `c6.4xlarge` da AWS por dia da semana, normalizado pelo preço no plano sob demanda.

As Figuras 2 e 3 acima apontam, entretanto, que não há relação direta entre os preços cobrados e os dias da semana e horas do dia, respectivamente. Logo, esse agendamento estático não traria o controle esperado para gastos e disponibilidade. Para alcançar esse controle é necessário decidir dinamicamente quando alocar uma instância baseado em seu preço atual. Essa estratégia deve ser aliada ao monitoramento de carga de maneira a evitar alocação desnecessária, e permitir economizar custos e satisfazer QoS.

2.2. Caso de uso: monitoramento não intrusivo de carga

Uma aplicação que se beneficiaria grandemente de computação oportunística na nuvem e que também precisa ser protegida é o NILM (*Non-Intrusive Load Monitoring*, i.e. monitoramento não intrusivo de carga) ou NIALM (*Non-Intrusive Appliance Load Monitoring*, i.e. monitoramento não intrusivo da carga de aparelhos [Hart 1992]). NILM é um processo para deduzir quais aparelhos foram utilizados em um dado instante, bem como desagregar seus consumos individuais, a partir de sinais energéticos simples de um edifício. O uso de técnicas como o NIALM para prover informações ao usuário está relacionado com economias de mais de 12% no uso de energia elétrica [Carrie Armel et al. 2013].

O caso de uso escolhido para esse trabalho é uma implementação NIALM da LiteMe, uma empresa de inteligência energética, que utiliza Redes Neurais Convolucionais para desagregar dados [LiteMe 2022]. Componentes auxiliares produzem itens de trabalho e os publicam em uma fila de mensagens para que o componente de NIALM possa trabalhar. Cada item de trabalho possui todas as informações necessárias para executar

um passo de desagregação (e.g. o medidor utilizado, os dados de energia, a topologia da rede, etc.). O conjunto desses itens na fila compõe a carga de trabalho a ser processada. Múltiplos contêineres de NIALM podem ser paralelizadas para processar mais carga se necessário.

LiteMe utiliza os resultados desagregados para prover serviços de inteligência energética a seus clientes, tais como monitoramento e previsões de contas. Clientes da LiteMe acessam serviços relacionados a aparelhos apenas esporadicamente ao longo do dia, o que permite atrasos no processamento dos trabalhos de desagregação. Além disso, para melhorar a acurácia da desagregação não são exibidos dados instantâneos de desagregação, mas apenas na granularidade de hora ou dia. Entretanto, se o processamento for atrasado demais, clientes podem sofrer inconveniências, o que fere termos de QoS.

3. Arquitetura usando VMs oportunistas para processamento em tempo quase real

A solução proposta para contornar os problemas apontados no uso de instâncias oportunistas consiste em monitorar tanto os custos dessas VMs, quanto a fila de trabalho, a fim de decidir se a aplicação deve processar a carga, e a replicação adequada para isso, ou se seria melhor esperar algum tempo até que o preço das máquinas diminuam. A arquitetura elaborada conta com três componentes: O JobMonitor, responsável por informar se o atraso no processamento está comprometendo a QoS da aplicação; o SpotSaver, que fornece informações sobre o preço que está sendo cobrado pelas instâncias; e o WorkWise, que consulta os outros dois serviços via API REST e a partir das informações obtidas calcula e escalona a quantidade de réplicas (i.e. *Pods*) necessárias no cluster Kubernetes (bem como os nós para executá-las). Essa arquitetura permite fácil evolução de cada um dos três módulos, explicados em detalhes abaixo.

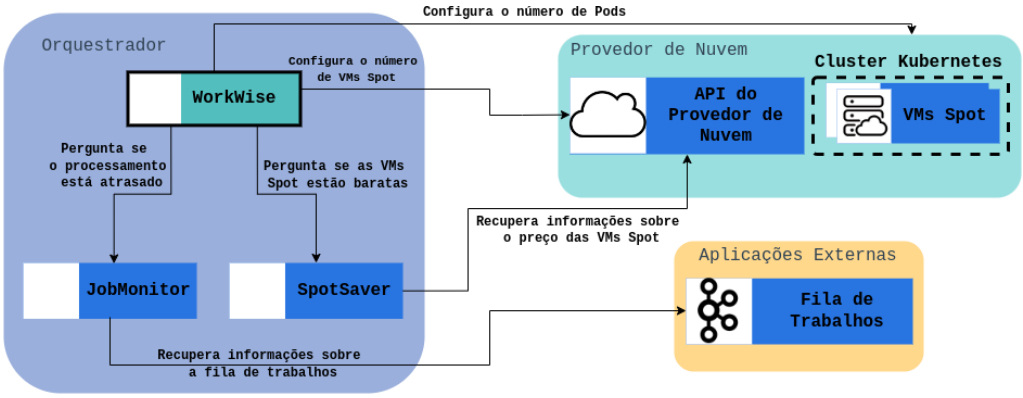

Figura 4. Diagrama da arquitetura proposta.

O JobMonitor requisita à fila de mensagens quantos trabalhos restam a ser processados e o momento da criação do trabalho pendente mais antigo. Com essas informações, o serviço é capaz de informar o quão atrasado está o processamento baseado no número de trabalhos restantes e há quanto tempo o trabalho mais antigo foi criado, além de se esse atraso comprometerá a QoS da aplicação que está sendo escalonada. Isso é decidido ao comparar com a tolerância máxima de atraso especificada.

O SpotSaver requisita ao provedor de nuvem informações sobre as instâncias, como o custo da VM no plano oportunista e no plano sob demanda, em cada zona de disponibilidade da região de interesse. A partir desse retorno, verifica se o custo da VM oportunista normalizado pelo custo no plano sob demanda está dentro do limite aceitável declarado. O serviço retorna o tipo da VM, a zona de disponibilidade, o seu preço atual, e se esse valor deve ser considerado barato ou não.

Por fim, o WorkWise consulta os dois outros serviços periodicamente para saber se a carga está atrasada e se o custo das instâncias está barato (i.e., dentro do aceitável). Com base nisso, ele calcula quantas réplicas da aplicação são necessárias, e quantas instâncias devem ser alocadas para executá-las.

Inicialmente o WorkWise verifica se o preço atual é considerado barato. Caso esteja barato, ele define a quantidade de trabalhos a serem processados igual ao tamanho da fila de trabalho. Caso o custo não seja aceitável, mas o atraso permaneça tolerável, as instâncias são desalocadas e todas as réplicas são removidas. Caso o valor esteja caro e o atraso seja crítico, o serviço vai atuar em controle de danos. Isto é, será alocada a quantidade mínima possível de instâncias que suporte replicação suficiente para aliviar o atraso por ao menos mais um ciclo. Isso permite que haja espaço para checar o preço novamente, e verificar se a situação já melhorou. Se durante o controle de danos, instâncias oportunistas não estejam disponíveis, são alocadas instâncias sob demanda de forma emergencial.

Para calcular quantas réplicas são necessárias para processar uma determinada carga, o serviço considera o tempo médio para cada trabalho ser finalizado pela aplicação. Outra informação necessária é a quantidade de réplicas que uma instância alocada suporta. O WorkWise utiliza isso para otimizar o uso de cada instância. A Figura 4 ilustra a arquitetura geral do sistema, assim como seus subcomponentes.

A implementação dos componentes é simples em seu estado atual, e a arquitetura é intencionalmente modular, tal que seja encorajada posterior sofisticação de um ou mais componentes.

4. Avaliação

Para avaliar a eficiência da solução proposta, foram simulados os custos com a infraestrutura do NIALM utilizando instâncias sob demanda, reservadas e oportunistas (com e sem ajuda do escalonador). A capacidade do NIALM é de processar um trabalho a cada 1.5 segundos [GAMA 2022], e foi considerada para ele uma carga de cerca de 20 trabalhos a cada segundo, o que seria equivalente a 1200 sensores monitorados, uma projeção futura de carga aceitável para a empresa. Assim, seriam necessárias cerca de 30 réplicas do NIALM em funcionamento constante para suportar uma demanda contínua. O NIALM foi executado em Kubernetes como um *Deployment* com a imagem Docker do módulo de desagregação, e replicado ao configurar o número de *Pods* correspondente àquele *Deployment*. A instância escolhida para o experimento foi a `c6g.4xlarge` da AWS, por ser capaz de executar 30 réplicas do NIALM. Para todos os cenários, foram calculados os custos diários para suprir a demanda do NIALM.

Uma vez que o tamanho de VM escolhido é capaz de processar a carga sem atraso, é possível considerar o custo de um dia para planos sem oscilações de preços (instâncias reservadas e sob demanda) multiplicando o preço horário ao longo do dia. Já para calcular o custo de VMs oportunistas sem escalonador (i.e., alocando independentemente do

preço), foi considerado o somatório de cada valor registrado, multiplicado pelo tempo em que aquele valor estava vigente.

Por fim, para calcular o custo com uso do escalonador, a carga foi preparada de maneira mais realista utilizando dados sintéticos aceitos pelo NIALM. O preço considerado aceitável foi definido como 65% do preço da instância sob demanda, e a tolerância máxima de atraso na carga foi de 3 horas. A cada ciclo de 10 minutos foi checada a decisão que o escalonador tomaria a partir do preço da instância naquele momento, bem como pelo o atraso da carga de trabalho. Foram registradas quantas instâncias seriam usadas e seus preços, para então calcular o custo somando os valores que seriam gastos em cada ciclo.

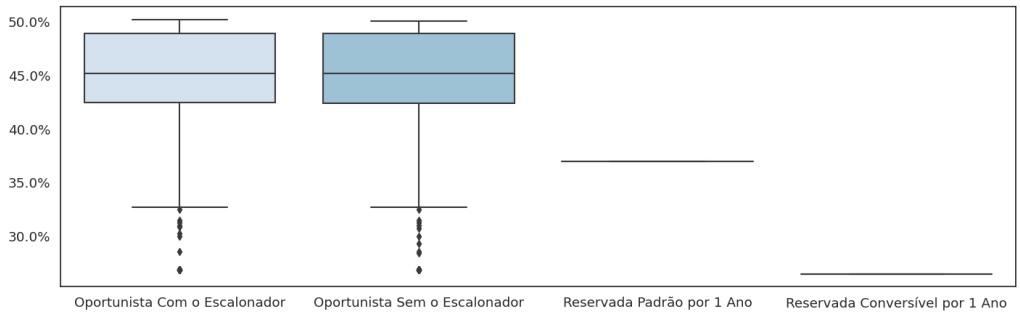

Figura 5. Economia diária em relação ao uso de instâncias no plano sob demanda

Para realizar os experimentos foram escolhidos os preços do ano de 2021 [Ardi 2022], por serem os dados mais recentes, que representam um ano completo, no momento em que as analises iniciaram. A Figura 5 acima ilustra a economia diária do uso da instância em cada plano, em relação ao preço no plano sob demanda (i.e., o custo diário da modalidade normalizado pelo custo do plano sob demanda, em formato de porcentagem). Uma vez que os preços das instancias reservadas são constantes durante todo o contrato, elas apresentaram um único valor diário durante todo o ano. Para as reservadas no plano do tipo padrão a economia foi de cerca de 37%, e para as no plano conversível foi próxima a 26% . Em média o custo do uso diário de instâncias oportunistas (independente do uso do escalonador) se mostrou consideravelmente menor do que o uso dos planos de instâncias reservadas. Os gastos para uso de VMs Spot com e sem o escalonador se mostraram praticamente iguais, com a maioria dos valores de economia diária variando entre 42% e 48%, mas alcançando até 50%. Todavia é importante relembrar que utilizar apenas as instâncias oportunistas não traz a garantia de disponibilidade desejada, problema esse que é contornado pelo o escalonador.

5. Conclusão

O uso de instâncias oportunistas impõe desafios e restrições no tipo de QoS que uma aplicação hospedada pode oferecer. Foi possível concluir que o uso do escalonador permitiu diminuir essas restrições, tal que aplicações com processamento em tempo quase real possam usufruir dos descontos dessas instâncias, sem ferir termos de QoS. Além disso, durante a simulação foi possível observar uma economia significativa de até 50% quando comparado com uso de instâncias sob demanda.

Implementações mais sofisticadas para os serviços JobMonitor e SpotSaver podem otimizar a capacidade do escalonador de reduzir custos, potencialmente o deixando ainda mais econômico. Por fim, trabalhos futuros que aliem segurança de cadeias de suprimento de *software* ao escalonador podem incentivar ainda mais a migração de aplicações de dados sensíveis como NILM/NIALM para a nuvem.

6. Agradecimentos

Este trabalho descreve o TCC (Trabalho de Conclusão de Curso, em fase de finalização) de Caetano Albuquerque, cujo foco é o uso de recursos oportunistas para o NIALM. Este trabalho é baseado no TCC de Diego Gama (concluído em 2022) que sugeriu uma abordagem baseada em tarefas para a execução do NIALM. Nós agradecemos a LiteMe pelo apoio (em colaboração com o núcleo UFCG/CEEI/EMBRAPII), em especial pelas informações sobre o caso de uso e pela disponibilização de dados para validação.

Referências

Ardi, C. (2022). Amazon EC2 Spot Price History: 2014–2015, 2017–2021. `https://ant.isi.edu/~calvin/data/ec2-spot-price/`.

Carrie Armel, K., Gupta, A., Shrimali, G., and Albert, A. (2013). Is disaggregation the holy grail of energy efficiency? the case of electricity. *Energy Policy*, 52:213–234. Special Section: Transition Pathways to a Low Carbon Economy.

GAMA, D. A. (2022). Desagregação distribuída: evolução arquitetural do desagregador nialm da liteme.

Hart, G. (1992). Nonintrusive appliance load monitoring. *Proceedings of the IEEE*, 80(12):1870–1891.

Kumar, D. P., Baranwal, G., Raza, Z., and Vidyarthi, D. P. (2017). A survey on spot pricing in cloud computing. *Journal of Network and Systems Management*, 26:809 – 856.

LiteMe (2022). LiteMe — Inteligência Energética. `https://liteme.com.br/about`.

Okafor, C., Schorlemmer, T. R., Torres-Arias, S., and Davis, J. C. (2022). Sok: Analysis of software supply chain security by establishing secure design properties. In *Proceedings of the 2022 ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*, SCORED'22, page 15–24, New York, NY, USA. Association for Computing Machinery.

Peisert, S., Schneier, B., Okhravi, H., Massacci, F., Benzel, T., Landwehr, C., Mannan, M., Mirkovic, J., Prakash, A., and Michael, J. B. (2021). Perspectives on the solarwinds incident. *IEEE Security & Privacy*, 19(2):7–13.

Services, A. W. (2022). Instâncias spot do Amazon EC2. `https://aws.amazon.com/pt/ec2/spot/`.

Tamrakar, K., Yazidi, A., and Haugerud, H. (2017). Cost efficient batch processing in amazon cloud with deadline awareness. In *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, pages 963–971.