

COMPUTAÇÃO DE ALTO DESEMPENHO RESISTENTE A FALHA EM AMBIENTE DE NUVEM

Universidade Presbiteriana Mackenzie

Coordenadoria de Fomento à Pesquisa - Pró-reitoria de Pesquisa e Pós-Graduação

Jessica dos Santos Santana Bispo¹

Apoio: PIBIC Mackenzie

Calebe de Paula Bianchini²

Faculdade de Computação e Informática^{1,2}

Programa de Pós-graduação em Computação Aplicada – PPGCA²

Universidade Presbiteriana Mackenzie – São Paulo, SP – Brasil

`jessica.bispo@mackenzista.com.br`¹

`calebe.bianchini@mackenzie.br`²

RESUMO

O crescimento exponencial da demanda por recursos computacionais nas últimas três décadas, impulsionado especialmente pela inteligência artificial, tem tornado crítica a necessidade de otimização em ambientes de computação de alto desempenho (HPC). Este trabalho analisa a eficiência de mecanismos de tolerância a falhas em aplicações HPC executadas em instâncias spot de provedores de nuvem, com foco na biblioteca DeLIA (Dependability Library for Iterative Applications). O estudo propõe uma transição do paradigma de polling, atualmente implementado no DeLIA, para uma arquitetura baseada em publish-subscribe para detecção de eventos de terminação de instâncias. A abordagem atual utiliza consultas periódicas a cada 5 segundos aos metadados da instância, gerando aproximadamente 8.640 requisições HTTP desnecessárias em uma execução de 12 horas. A solução proposta implementa um endpoint HTTP que recebe notificações proativas dos provedores de nuvem através de seus gerenciadores de eventos (AWS EventBridge, Azure Event Grid, Google Cloud Pub/Sub), eliminando consultas redundantes e reduzindo significativamente o overhead computacional. A arquitetura event-driven mantém compatibilidade com múltiplos provedores de nuvem através de uma interface agnóstica baseada em requisições POST. Os resultados esperados incluem maior eficiência no uso de recursos, menor latência de resposta na detecção de terminações e melhor escalabilidade para aplicações HPC em ambientes de nuvem, contribuindo para o desenvolvimento de sistemas mais sustentáveis e economicamente viáveis.

Palavras-chave: Computação de Alto Desempenho. Tolerância a Falhas. Publish-Subscribe. Instâncias Spot. Otimização Computacional.

ABSTRACT

The exponential growth in demand for computing resources over the last three decades, driven by artificial intelligence, has made optimization in high-performance computing (HPC) environments critical. This work analyzes the efficiency of fault tolerance mechanisms in HPC applications running on cloud providers' spot instances, focusing on the DeLIA (Dependability Library for Iterative Applications) library. The study proposes a transition from the polling paradigm, currently implemented in DeLIA, to a publish-subscribe based architecture for detecting instance termination events. The current approach uses periodic queries every 5 seconds to the instance metadata, generating approximately 8,640 unnecessary HTTP requests in a 12-hour execution. The proposed solution implements an HTTP endpoint that receives

proactive notifications from cloud providers through their event managers (AWS EventBridge, Azure Event Grid, Google Cloud Pub/Sub), eliminating redundant queries and significantly reducing computation overhead. The event-driven architecture maintains compatibility with multiple cloud providers through an agnostic interface based on POST requests. Expected results include greater resource efficiency, lower response latency in termination detection, and improved scalability for HPC applications in cloud environments, contributing to the development of more sustainable and economically viable systems.

Keywords: High-Performance Computing. Fault Tolerance. Publish-Subscribe. Spot Instances. Computational Optimization.

1. INTRODUÇÃO

Quando observamos o passado recente e a evolução da utilização de recursos computacionais, identificamos uma curva de crescimento que aumentou exponencialmente nas últimas três décadas. O avanço tecnológico trouxe consigo uma demanda crescente por processamento, armazenamento e transmissão de dados — uma demanda que, muitas vezes, não percebemos no cotidiano. Por exemplo, quando assistimos a um vídeo via celular, podemos consumir até 100 vezes mais energia (do que gastamos em nosso aparelho) no "invisível": o caminho que o dado percorre até chegar até nós (INTERNATIONAL ENERGY AGENCY (IEA), 2024).

Seja em atividades mundanas como assistir a programas em streaming, navegar em lojas online no notebook ou fazer videochamadas, até tarefas altamente demandantes de recursos como pesquisas científicas em ambientes de alto desempenho (*High-Performance Computing* – HPC) e treinamentos de grandes modelos de inteligência artificial, a quantidade de energia e capacidade computacional necessária continua crescendo.

Quando retornamos ao primeiro computador eletrônico, o ENIAC, construído na Universidade de Pennsylvania em 1945, este consumia aproximadamente 150kW — na época, a mesma quantidade de energia que 100 dormitórios utilizavam para se manter operantes (BRITANNICA, 2008). Hoje, podemos observar uma grande mudança nessa escala: data centers modernos operam com demandas energéticas da ordem de megawatts (INTERNATIONAL ENERGY AGENCY (IEA), 2024), e a rede global de computadores e dispositivos já representa uma parcela significativa do consumo energético mundial.

A proporção de energia consumida por computação — inferior a 2% em 2000 — ultrapassou 6% em 2012 e pode chegar a 20% até 2025 (INTERNATIONAL ENERGY AGENCY (IEA), 2012). Na era da inteligência artificial, a exigência por poder de processamento intensificou-se: desde 2010, o “*compute*” necessário para treinar modelos dobra a cada seis meses (JAIME SEVILLE, 2022).

Esse cenário de crescimento acelerado no consumo de recursos computacionais, aliado à complexidade crescente das aplicações modernas, torna evidente a necessidade de soluções mais eficientes e resilientes, especialmente em ambientes de alto desempenho.

1.1. Contexto

Nesse contexto, buscamos, nesta pesquisa, realizar uma análise bibliográfica de aplicações com alto custo computacional que contam com resistência a falhas (em essência ou adição de bibliotecas complementares) para compreender pontos de otimização e melhoria de eficiência no uso de recursos computacionais.

Para o desenvolvimento deste estudo, analisamos o Mamute (FERNANDES et al., 2025), um software que oferece métodos geofísicos baseados na equação de onda acústica. Devido ao seu alto custo computacional, aplicações geofísicas são geralmente projetadas para execução em grandes sistemas de computação. Por isso, essas aplicações devem implementar diversas técnicas de alto desempenho para melhor utilizar os recursos computacionais. Uma dessas técnicas utiliza mecanismos de tolerância a falhas, incorporados por meio da biblioteca DeLIA.

Em sua aplicação, o DeLIA oferece alguns mecanismos importantes de tolerância a falhas, como: (i) monitoramento regular do estado da aplicação para detectar falhas e/ou disponibilidade (*heartbeat*); (ii) monitoramento de sinais de finalização e/ou término abrupto da aplicação; (iii) mecanismo de armazenamento (local e global) do estado da aplicação; (iv) técnicas de replicação do estado da aplicação; e (v) mecanismo de reinício da aplicação a partir de um estado estável (*rollback*).

O DeLIA, dentre outras aplicabilidades, foi desenvolvido para lidar com cenários preemptivos, capturando o sinal de terminação e desencadeando um processo de salvamento de dados locais. Atualmente, a captura desse sinal dentro do DeLIA, para esse cenário específico, é realizada por um mecanismo de *polling*. A cada 5 segundos, as informações do metadado da máquina virtual são buscadas para verificar se o atributo “*instance-action*” foi alterado para “*terminate*”, indicando que uma instância preemptiva será terminada em menos de 2 minutos (AWS, 2025).

1.2. Computação em Nuvem, instâncias *Spot* na AWS e seus desafios

Quando pensamos em melhor uso de recursos computacionais, a computação em nuvem geralmente surge como uma das primeiras opções. Quando pensamos mais especificamente em HPC, percebemos como uma alternativa muito viável a computação em nuvem, permitindo a

execução de tarefas científicas e experimentos que exigem alto poder de processamento (CORDEIRO et al, 2023).

Diferente das infraestruturas tradicionais, a nuvem possibilita a alocação dinâmica de recursos, oferecendo flexibilidade e escalabilidade sob um modelo de pagamento conforme o uso (SAMPAIO; BARBOSA, 2020). Nesse cenário, a AWS disponibiliza instâncias denominadas *spot* que permitem acessar recursos ociosos a um custo reduzido, embora sem garantia de disponibilidade, pois podem ser interrompidas a qualquer momento com um aviso prévio mínimo.

Apesar das vantagens econômicas, a volatilidade das instâncias *spot* representa um desafio para aplicações que exigem alta disponibilidade, tornando essencial o desenvolvimento de mecanismos de tolerância a falhas (SANTANA et al., 2024). Projetar ambientes resilientes para HPC em nuvem requer estratégias eficientes que garantam a continuidade das operações, mesmo diante de interrupções inesperadas (TANENBAUM; VAN STEEN, 2007).

1.3. Desafios da Tolerância a Falhas em HPC

Projetar ambientes resilientes em nuvem é um grande desafio, pois dada a complexidade da arquitetura, falhas são inevitáveis, foi percebido que um sistema com 100.000 processadores experienciavam falha a cada poucos minutos (ENGELMANN AND GEIST, 2005), tal nível de complexidade requer o desenvolvimento de estratégias sofisticadas e eficientes que garantam a continuidade das operações computacionais, mesmo diante de interrupções inesperadas e potencialmente frequentes (TANENBAUM; VAN STEEN, 2007). Essas estratégias devem considerar não apenas a detecção rápida e confiável de falhas, mas também mecanismos eficazes de recuperação, redistribuição de cargas de trabalho e preservação do estado da aplicação.

1.4. A Biblioteca DeLIA

Neste contexto de necessidades específicas surge o DeLIA (*Dependability Library for Iterative Applications*), uma biblioteca especializada que implementa mecanismos abrangentes de confiabilidade para aplicações iterativas. Esta biblioteca foi especificamente projetada para programas que utilizam infraestruturas de HPC e que exigem sincronização rigorosa de dados entre todos os processos participantes da computação distribuída.

A biblioteca DeLIA oferece funcionalidades críticas para a manutenção da integridade computacional, incluindo o armazenamento sistemático do estado global da aplicação a cada iteração do algoritmo. Para garantir essa funcionalidade, a DeLIA implementa mecanismos de *checkpointing* (criação de pontos de verificação) e *rollback* (retorno a estados anteriores em bom estado) tanto do estado global da aplicação quanto do estado local de cada processo individual participante da computação distribuída.

1.5. Detecção de Interrupções por *Pooling*

Para viabilizar esses mecanismos de proteção, a biblioteca DeLIA incorpora sistemas de detecção de interrupções e falhas que são fundamentais para o seu funcionamento. Especificamente no contexto do uso de instâncias spot na AWS, a versão atual da DeLIA implementa a detecção de interrupção de instâncias por meio de uma técnica conhecida como *polling*, uma abordagem que consiste na consulta periódica e regular aos metadados da instância para verificar se foi emitido um sinal de terminação (SANTANA et al., 2024).

1.6. Proposta de Melhoria

Embora funcione bem, a abordagem atual de *polling* apresenta limitações significativas em termos de eficiência de recursos, escalabilidade e latência de resposta. Assim, este trabalho apresenta uma versão alternativa e potencialmente superior para detectar as notificações das provedoras de cloud sobre o aviso de encerramento das instâncias spot, utilizando uma arquitetura baseada no paradigma *publish-subscribe*.

1.7. Objetivos do Trabalho

Com a implementação desta nova arquitetura, espera-se avaliar se esta versão oferece maior eficiência computacional e menor sobrecarga de recursos de sistema. Adicionalmente, objetiva-se aumentar significativamente a escalabilidade de uso de instâncias spot por aplicações HPC sem sobrecarregar o ambiente de nuvem com consultas desnecessárias e repetitivas sobre o status de encerramento das instâncias (VAN DE VYVERE; COLPAERT; VERBORGH, 2020; TANENBAUM; VAN STEEN, 2007).

2. DESENVOLVIMENTO DO ARGUMENTO

Após intensa análise em toda a arquitetura, decidimos utilizar o DeLIA, como caso de teste para esse estudo. O DeLIA é uma biblioteca que combina preservação de dados, capacidade de *failover* e detecção de interrupção utilizada para adicionar resiliência para aplicações que utilizam recursos computacionais intensivamente, como aplicações de alto desempenho.

O DeLIA tem como objetivo central o tratamento de falhas do tipo *fail-stop*, que se caracterizam por interrupções abruptas no processamento da aplicação. Essas interrupções podem resultar tanto de defeitos de hardware quanto de falhas específicas de software, provocando o encerramento inesperado da aplicação sem a possibilidade de recuperação automática (HERAULT; ROBERT, 2015).

Para atingir esse objetivo, a biblioteca implementa uma arquitetura robusta baseada em três pilares fundamentais: detecção de interrupções, *checkpointing* e recuperação por *rollback*. O sistema de detecção opera através de duas abordagens complementares: monitoramento por *heartbeat* e captura de sinais de terminação. No monitoramento por *heartbeat*, um nó líder atua como detector de falhas, recebendo mensagens periódicas dos demais nós em intervalos configuráveis. Caso não receba uma mensagem dentro do tempo máximo esperado, assume que o nó falhou e dispara o processo de salvamento de dados locais em todos os outros nós.

Paralelamente, o DeLIA monitora sinais de terminação enviados por sistemas em nuvem. Esses sinais indicam que uma aplicação deve ser encerrada por razões como falhas do sistema, excesso do tempo alocado ou recuperação de recursos em ambientes preemptivos. Quando detecta esses sinais, a biblioteca imediatamente aciona o mecanismo de preservação de dados, garantindo que o progresso da aplicação seja mantido mesmo em caso de terminação forçada.

O sistema de *checkpointing* do DeLIA permite que os desenvolvedores definam quais dados críticos devem ser salvos e com que frequência, considerando que o *overhead* de salvamento deve ser menor que o tempo de recomputação.

Além disso, o sistema foi desenvolvido para gerenciar situações preemptivas, definidas como contextos nos quais a execução da aplicação é interrompida sem que haja necessariamente uma falha subjacente.

Para ambientes preemptivos, como um ambiente de nuvem, o DeLIA realiza *polling* do metadado da instância, utilizando o *script* apresentado na Figura 1 (AWS, 2025).

```
#!/bin/bash

# Your application execution with DeLIA
mpirun -n <N> <YOUR APPLICATION NAME>

TOKEN=`curl -s -X PUT "http://169.254.169.254/latest/api/token" -H "X-aws-ec2-metadata-token-ttl-seconds: 21600"`

while sleep 5; do

    HTTP_CODE=$(curl -H "X-aws-ec2-metadata-token: $TOKEN" -s -w %{http_code} -o /dev/null
        http://169.254.169.254/latest/meta-data/spot/instance-action)

    if [[ "$HTTP_CODE" -eq 401 ]] ; then
        echo 'Refreshing Authentication Token'
        TOKEN=`curl -s -X PUT "http://169.254.169.254/latest/api/token" -H "X-aws-ec2-metadata-token-ttl-seconds: 30"`
    elif [[ "$HTTP_CODE" -eq 200 ]] ; then
        echo 'Detected interrupt signal'
        # Send SIGTERM
        killall -s 15 <YOUR APPLICATION NAME>
    else
        echo 'Not Interrupted'
    fi
done
```

Figura 1. Script de *pooling*, baseado na recomendação da (AWS, 2025).

Apesar de uma abordagem funcional, buscamos entender se alterar o comportamento da verificação por *pooling* para *publish-subscribe* seria interessante para um ambiente de cloud, buscando adicionar uma cama de orientação a evento ao DeLIA.

O paradigma de *pooling*, atualmente implementado no DeLIA, envolve clientes repetidamente requisitando (*pooling*) um servidor por atualizações, é como se, ao fazer um pedido em um aplicativo de delivery, você ficasse o tempo todo perguntando no aplicativo “o pedido chegou?”, é possível que em algum momento você consiga a resposta que deseja, mas é um tanto ineficiente.

Já o paradigma de *Publish-Subscribe* envolve uma abordagem diferente: em vez de o cliente ficar perguntando constantemente, ele simplesmente "se inscreve" para receber uma notificação quando algo interessante acontecer, como a entrega do pedido. Continuando com a analogia do delivery, seria como se você deixasse seu telefone disponível e o aplicativo avisasse quando o pedido chegasse ao endereço de entrega.

No contexto técnico, quando uma aplicação utiliza *publish-subscribe*, ela registra seu interesse em eventos específicos (o "*subscribe*") junto ao provedor de serviço (no contexto do DeLIA, o interesse de receber eventos relacionados a terminação da máquina). Quando esses eventos ocorrem, o provedor automaticamente envia uma notificação para todos os interessados (o "*publish*"). Isso elimina a necessidade de consultas repetitivas e desnecessárias, resultando em menor consumo de recursos computacionais e de rede.

O foco deste projeto foi na análise de uma solução que adicionasse ao funcionamento do DeLIA uma funcionalidade que pudesse detectar e reagir à notificação de encerramento de instâncias *spot* de forma eficiente, garantindo que os processos em execução fossem concluídos ou migrados sem perdas significativas de dados. Esperando-se alterar o modelo atual de *polling* para um modelo mais escalável, como o *publish-subscribe* [TANENBAUM; VAN STEEN, 2007).

Primeiramente, foi estudado o mecanismo de envio do aviso de interrupção de máquinas *preemptivas* pela AWS: Dado o amplo espectro de provedores de nuvem, fez-se necessário compreender as particularidades dos avisos de interrupção das máquinas *preemptivas*, pois cada provedor possui abordagens distintas, por exemplo:

- na AWS, você pode realizar um *polling* no metadado da instância para verificar se a instância está para ser terminada, com um tempo de aviso para o término de 2 minutos. Outra forma de receber o aviso pode ser através da plataforma de eventos da AWS, o EventBridge;
- no caso da Azure, o tempo de aviso é de 30 segundos e pode ser capturado via metadados da instância. A Azure oferece o Azure Event Grid como serviço de gerenciamento de eventos para capturar avisos de terminação; e
- no Google Cloud Platform (GCP), pode haver um aviso de terminação, mas sem garantia de sua disponibilização. O Google Cloud Pub/Sub e o Cloud Eventarc podem ser utilizados para gerenciar eventos de terminação de instâncias.

Dada a compreensão de que existe pelo menos dois modos de receber esse aviso de terminação, é necessário realizar uma comparação e entender qual teria um melhor custo-benefício: (i) realizar o *polling* da informação (ii) utilizar o gerenciador de eventos próprio da provedora (na AWS é o EventBridge) (iii) utilizar o gerenciador de eventos em conjunto com o *polling* com uma menor frequência.

Ao realizar o *polling* da informação, usando o tempo definido no DeLIA de 5 segundos, além dos recursos da máquina na execução dos *scripts bash*, como requisição de rede e ciclos da CPU, há o envio de diversas requisições HTTP REST mesmo que não exista o registro de terminação da instância. Considerando uma instância *spot* que foi utilizada durante 12 horas antes do aviso de seu término, há aproximadamente 8.640 (720 requisições por hora * 12 horas) requisições HTTP REST desnecessárias enviadas a partir da instância *spot*.

Ao utilizar um gerenciador de eventos, na AWS temos a opção do EventBridge. O EventBridge opera de forma assíncrona e sob demanda, incluindo dois modos de processar eventos: *event buses*, que são roteadores que recebem um evento e entregam para 0 ou mais alvos, e *pipes*, que realizam integrações ponta-a-ponta, onde cada *pipe* recebe eventos de uma única fonte de processamento e entrega para um único alvo. *Event buses* e *pipes* podem ser utilizados em conjunto, sendo um caso comum a criação de um *pipe* que tenha como alvo um *event bus*, este enviando o evento para diversos alvos (AWS, 2025).

Um gerenciador de eventos pode aumentar o grau de abstração da aplicação, pois traz flexibilidade sobre o tratamento dos eventos de terminação de instância. Além da possibilidade de emitir um evento indicando a terminação da aplicação (que eventualmente será *graceful*), o gerenciador de eventos permite acionar outras ações, como, por exemplo, a possibilidade do armazenamento de dados.

A solução adotada para o protótipo considera o fluxo determinado pela AWS, em que um evento de término enviado pelo EC2 para uma instância *spot i* qualquer será capturada pelo EventBridge. Este, por sua vez, notificará a aplicação, por meio da arquitetura *publish-subscribe*, que ela deverá finalizar sua computação. A aplicação captura essa notificação devido a nova arquitetura de mensagens proposta para o DeLIA. A Figura 2 mostra as principais mensagens enviadas na arquitetura da solução proposta utilizando os componentes da AWS: o EC2 cria o evento de terminação e o envia ao Event Bridge, que irá notificar a aplicação para iniciar o processo de terminação (1a) e notifica a máquina *spot* 4 de seu término (1b.).

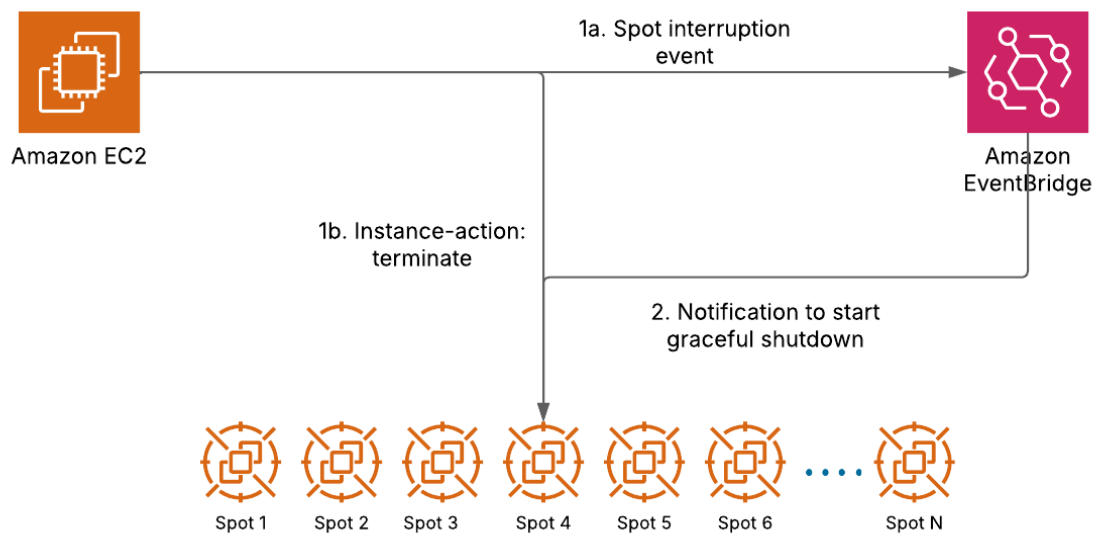


Figura 2. Nova arquitetura proposta usando componentes da AWS.

3. CONCLUSÃO

Este projeto investigou a otimização de mecanismos de tolerância a falhas em aplicações de computação de alto desempenho (HPC) executadas em ambientes de nuvem, especificamente focando na transição do paradigma de *pooling* para *publish-subscribe* na biblioteca *DeLLIA*. O problema de pesquisa centrou-se na identificação de ineficiências no modelo atual de detecção de terminação de instâncias spot, que gera overhead computacional significativo através de consultas repetitivas e desnecessárias aos metadados do provedor de nuvem.

O objetivo principal foi desenvolver e avaliar uma arquitetura *event-driven* alternativa que pudesse reduzir o consumo de recursos computacionais mantendo a eficácia na detecção de eventos de terminação, para tal, utilizou-se uma metodologia de análise bibliográfica

A principal contribuição desta pesquisa reside na proposta de uma solução agnóstica aos provedores de nuvem que mantém compatibilidade com os principais serviços de gerenciamento de eventos (*AWS EventBridge*, *Azure Event Grid*, *Google Cloud Pub/Sub*) através de uma interface HTTP padronizada. Esta abordagem permite que aplicações HPC se beneficiem da otimização proposta independentemente do provedor escolhido, aumentando a portabilidade e flexibilidade das soluções.

No entanto, algumas limitações devem ser reconhecidas. O estudo focou especificamente em instâncias spot e cenários preemptivos, não abordando outros tipos de

falhas ou ambientes de HPC tradicionais. Além disso, a validação experimental limitou-se ao contexto da AWS, embora a arquitetura tenha sido projetada para múltiplos provedores. A dependência de conectividade de rede para recebimento de eventos também representa um potencial vulnerabilidade que requer investigação adicional.

Como trabalhos futuros, sugere-se a implementação e teste da solução vários provedores de nuvem para validar a portabilidade da proposta, além do mais, uma avaliação quantitativa detalhada do impacto na performance de aplicações reais, incluindo medições de latência, *throughput* e consumo energético, seria valiosa para confirmar os benefícios teóricos identificados. Adicionalmente, a investigação profunda de mecanismos híbridos que combinem *publish-subscribe* com *polling* de baixa frequência como *fallback* poderia aumentar a robustez da solução.

Por fim, a aplicação dos princípios identificados neste trabalho a outras bibliotecas de tolerância a falhas poderia amplificar o impacto das otimizações propostas no ecossistema de HPC,

Essa proposta de solução também foi apresentada no ERAD-SP 2025, e publicada sob título “*Implementação da técnica de publish-subscribe para detectar uma interrupção de instâncias preemptivas*” nos Anais da XVI Escola Regional de Alto Desempenho de São Paulo (BISPO; BIANCHINI, 2025).

Os autores agradecem ao MackCloud, Laboratório Multidisciplinar de Computação Científica e Nuvem da Universidade Presbiteriana Mackenzie.

4. REFERÊNCIAS

AWS. *Spot Instance interruption notices*. In: *Amazon EC2 User Guide for Linux Instances*. [S.l.]. Disponível em: <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/spot-instancetermination-notice.html>. Acesso em: 31 mar. 2025.

BISPO, Jessica S. S.; BIANCHINI, Calebe P. Implementação da técnica de publish-subscribe para detectar uma interrupção de instâncias preemptivas. In: ESCOLA REGIONAL DE ALTO DESEMPENHO DE SÃO PAULO (ERAD-SP), 16., 2025, São José do Rio Preto/SP. *Anais [...]*. Porto Alegre: Sociedade Brasileira de Computação, 2025. p. 9-12. DOI: <https://doi.org/10.5753/eradsp.2025.9576>.

BRITANNICA. *ENIAC*. In: *Encyclopedia Britannica*. [S.l.]: Britannica, 2008. Disponível em: <https://www.britannica.com/technology/ENIAC>. Acesso em: 29 ago. 2025.

CORDEIRO, Daniel; et al. Green cloud computing: challenges and opportunities. In: SIMPÓSIO BRASILEIRO DE SISTEMAS DE INFORMAÇÃO (SBSI), 19., 2023, Maceió/AL. *Anais [...]*. Porto Alegre: Sociedade Brasileira de Computação, 2023. p. 129-131. DOI: https://doi.org/10.5753/sbsi_estendido.2023.229291.

DONGARRA, J.; HERAULT, T.; ROBERT, Y. Fault tolerance techniques for high-performance computing. In: HERAULT, T.; ROBERT, Y. (orgs.). *Fault-tolerance techniques for high-performance computing*. Cham: Springer, 2015. p. 3-85. (Computer Communications and Networks). Disponível em: https://doi.org/10.1007/978-3-319-20943-2_1. Acesso em: 29 ago. 2025.

ENERGY INFORMATION ADMINISTRATION (EIA). *Commercial Buildings Energy Consumption Survey (CBECS): 2012 Commercial Buildings Energy Consumption Survey: Energy Usage Summary*. Washington, DC: U.S. Department of Energy, 2016. Disponível em: <https://www.eia.gov/consumption/commercial/reports/2012/energyusage/>. Acesso em: 29 ago. 2025.

ENGELMANN, Christian; GEIST, Al. Super-scalable algorithms for computing on 100,000 processors. In: SUNDERAM, V. S.; VAN ALBADA, G. D.; SLOOT, P. M. A.; DONGARRA, J. J. (eds.). *Computational science – ICCS 2005. Lecture Notes in Computer Science*, v. 3514, n. 1, p. 313–321. Berlin; Heidelberg: Springer, 2005. Disponível em: https://doi.org/10.1007/11428831_39. Acesso em: 29 ago. 2025.

FERNANDES, João B.; et al. Mamute: high-performance computing for geophysical methods. *arXiv*, 17 fev. 2025. Disponível em: <https://doi.org/10.48550/arXiv.2502.12350>.

HWANG, Kai; et al. *Distributed and cloud computing: from parallel processing to the Internet of Things*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

INTERNATIONAL ENERGY AGENCY (IEA). *What the data centre and AI boom could mean for the energy sector*. Paris: IEA, 2024. Disponível em: <https://www.iea.org/commentaries/what-the-data-centre-and-ai-boom-could-mean-for-the-energy-sector>. Acesso em: 29 ago. 2025.

SAMPAIO, Altino M.; BARBOSA, Jorge G. Constructing reliable computing environments on top of Amazon EC2 Spot Instances. *Algorithms*, Suíça, p. 187-187, jul. 2020. DOI: <https://doi.org/10.3390/a13080187>.

SANTANA, Carla; et al. DeLIA: a dependability library for iterative applications applied to parallel geophysical problems. *Computers & Geosciences*, v. 191, p. 105662, 2024. DOI: <https://doi.org/10.1016/j.cageo.2024.105662>.

SEVILLA, Jaime; et al. Compute trends across three eras of machine learning. In: INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS (IJCNN), 2022, Padua, Italy. *Proceedings* [...]. Padua: IEEE, 2022. p. 1-8. Disponível em: <https://arxiv.org/pdf/2202.05924>. Acesso em: 29 ago. 2025.

TANENBAUM, A. S.; VAN STEEN, M. *Sistemas distribuídos: princípios e paradigmas*. 2. ed. São Paulo: Pearson Prentice Hall, 2007.

VAN DE VYVERE, Brecht; COLPAERT, Pieter; VERBORGH, Ruben. Comparing a polling and push-based approach for live open data interfaces. In: BIELIKOVA, Maria; MIKKONEN, Tommi; PAUTASSO, Cesare (eds.). *Web engineering*. Cham: Springer International Publishing, 2020. p. 87-101. ISBN 978-3-030-50578-3. DOI: https://doi.org/10.1007/978-3-030-50578-3_7.