Implementação da técnica de *publish-subscribe* para detectar uma interrupção de instâncias *preemptivas*

Jessica S. S. Bispo[1], Calebe P. Bianchini[1]

[1]Faculdade de Computação e Informação – Universidade Presbiteriana Mackenzie
São Paulo, SP - Brasil

jessica.bispo@mackenzista.com.br, calebe.bianchini@mackenzie.br

Resumo: *Este projeto apresenta uma abordagem event-driven para realizar a captura do sinal de terminação de máquinas preemptivas em ambiente de nuvem usando a técnica de publish-subscribe. Essa técnica foi projetada para o aprimoramento de uma biblioteca de tolerância a falhas para ambiente de alto desempenho oferecendo uma alternativa para detectar a finalização de uma máquina spot, alterando seu comportamento padrão de polling para essa nova técnica.*

1. Introdução

A crescente demanda por poder computacional impulsiona setores como inteligência artificial, análise de dados e *blockchain*, exigindo soluções eficientes para processar grandes volumes de dados. Nesse contexto, os supercomputadores e a computação de alto desempenho (*High-Performance Computing* – HPC) surgem como alternativas para lidar com problemas complexos, combinando múltiplos sistemas computacionais em arquiteturas diversas [HWANG et al., 2011].

A computação em nuvem tem se mostrado uma alternativa viável para HPC, 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 et al., 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].

O DeLIA[1] é uma biblioteca que implementa mecanismos de confiabilidade para aplicações iterativas projetada para programas que utilizam infraestruturas de HPC e que exigem sincronização de dados entre todos os processos. Essa biblioteca oferece um

[1] O DeLIA pode ser obtido em https://gitlab.com/lappsufrn/delia

armazenamento do estado global a cada iteração. Para isso, DeLIA implementa mecanismos de *checkpointing* e *rollback* tanto desse estado global quanto do estado local de cada processo. Para isso, o DeLIA incorpora mecanismos de detecção de interrupções ou falhas. No caso de uso de instâncias *spot* na AWS, o DeLIA implementa a detecção da interrupção da instância por meio de *polling* [SANTANA et al., 2024].

Assim, esse trabalho apresenta uma versão alternativa para detectar as notificações da AWS sobre o aviso de encerramento das instâncias *spot* usando uma arquitetura *publish-subscribe*. Espera-se, com isso, avaliar se essa nova versão oferece maior eficiência e baixa latência para atualizações enviadas em tempo real do ambiente de nuvem para o sistema de monitoramento do DeLIA. Espera-se também aumentar a escalabilidade de uso de instâncias *spot* pela aplicação HPC sem sobrecarregar o ambiente com consultas sobre a decisão de encerramento das instâncias *spot* [VAN DE VYVERE, 2020] [TANENBAUM; VAN STEEN, 2007].

2. Revisão Bibliográfica

Para o desenvolvimento desse estudo, utilizamos 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 serem executadas em grandes sistemas de computação. Por isso, essas aplicações devem implementar diversas técnicas de alto desempenho para utilizar melhor os recursos computacionais. Uma dessas técnicas utiliza-se de mecanismos de tolerância a falhas, incorporados a partir da biblioteca DeLIA.

Em sua aplicação, 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 desenhado para lidar com cenários *preemptivos*, captando o sinal de terminação e desencadeando um processo de salvamento de dados locais. A captura desse sinal dentro do DeLIA, para esse cenário específico, é feita 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].

3. Desenvolvimento

O foco deste projeto foi no desenvolvimento de uma solução 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. Além disso, esperava-se alterar o modelo atual de *polling* para outro 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 *scritps bash*, como requisição de rede e ciclos da CPU, há o envio de diversas requisições HTTP REST mesmo que não existe 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 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 1 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* 6 de seu término (1b.);

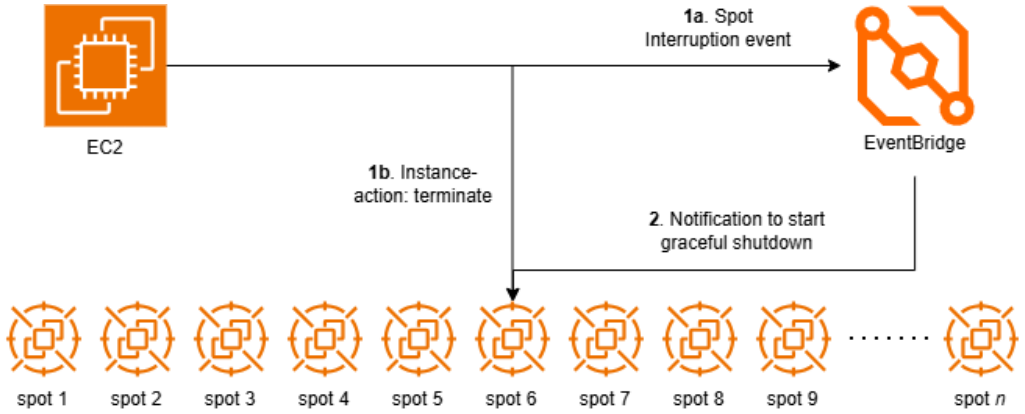

Figura 1. Nova arquitetura proposta usando componentes AWS.

4. Discussão

A proposta de melhoria da arquitetura do DeLIA apresentada neste artigo ainda está em fase de desenvolvimento e experimentação. Mesmo assim, a troca da arquitetura de *polling* para *publish-subscribe* diminuirá drasticamente a quantidade de mensagens trocadas no ambiente. Aplicando a proposta a arquitetura de componentes da AWS, é possível visualizar a praticidade da proposta.

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-instance-termination-notices.html. Acesso em: 31 mar. 2025.

CORDEIRO, Daniel; et al. Green Cloud Computing: Challenges and Opportunities. In: TRILHA DE TEMAS, IDEIAS E RESULTADOS EMERGENTES EM SISTEMAS DE INFORMAÇÃO - 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.

FERNANDES, João B.; et al. Mamute: high-performance computing for geophysical methods. 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.

SAMPAIO, Altino M.; BARBOSA, Jorge G. Constructing Reliable Computing Environments on Top of Amazon EC2 Spot Instances. *Algorithms*. Suiça, 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. Disponível. DOI: https://doi.org/10.1016/j.cageo.2024.105662

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 (Ed.). 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