Otimização de custo de um sistema de modelagem atmosférica em nuvens computacionais

Mateus S. de Melo[1], **Lúcia M. A. Drummond**[1], **Roberto P. Souto**[2]

[1]Instituto de Computação – Universidade Federal Fluminense (UFF)
Niterói, RJ

[2]Laboratório Nacional de Computação Científica
Petrópolis, RJ

msmelo@id.uff.br, lucia@ic.uff.br, rpsouto@lncc.br

Abstract. *This work proposes strategies to reduce the financial costs associated with running the BRAMS model on cloud-based clusters using AWS ParallelCluster. The first strategy employs low-cost instances (Spot), with a fault-tolerant version of BRAMS, allowing execution to resume from a checkpoint in case of revocation. The second strategy consists of an instance selection algorithm that compares costs across different regions and markets, recommending the most economical options, combined with a three-queue architecture to manage execution in case of revocation. The results for spatial domains of different sizes indicate that adopting a cloud-based cluster is a promising alternative for this High-Performance Computing application.*

Resumo. *Este trabalho propõe estratégias para reduzir os custos financeiros associados à execução do modelo BRAMS em clusters na nuvem, utilizando o AWS ParallelCluster. A primeira estratégia emprega instâncias de baixo custo (Spot), com uma versão tolerante a falhas do BRAMS, permitindo a retomada da execução a partir de um checkpoint em caso de revogação. A segunda estratégia consiste em um algoritmo de seleção de instâncias que compara custos em diferentes regiões e mercados, recomendando as opções mais econômicas, combinada com uma arquitetura de três filas para gerenciar a execução em caso de revogação. Os resultados para domínios espaciais de diferentes tamanhos indicam que a adoção de um cluster na nuvem é uma alternativa promissora para esta aplicação de Computação de Alto Desempenho.*

1. Introdução

O avanço da computação em nuvem tem aumentado o interesse em seu uso para aplicações de Computação de Alto Desempenho (HPC, do inglês *High Performance Computing*). Entretanto, apesar de vantagens como a escalabilidade sob demanda e a ausência de gestão de infraestrutura física, desafios como o alto custo e a limitação da rede persistem para aplicações fortemente acopladas, motivando a busca por estratégias de melhor custo-benefício para execução científica na nuvem. Na meteorologia, modelos numéricos de previsão do tempo e clima beneficiam-se da Computação de Alto Desempenho devido à necessidade de processar grandes volumes de dados e resolver equações complexas [Michalakes 2020]. A utilização de nuvens computacionais para essa finalidade tem sido investigada, com estudos avaliando sua viabilidade para diversos modelos

[Powers et al. 2021]. Este trabalho foca no BRAMS (*Brazilian developments on the Regional Atmospheric Modeling System*), um modelo numérico de previsão de tempo e clima em escala regional mantido pelo INPE/CPTEC (Instituto Nacional de Pesquisas Espaciais/Centro de Previsão de Tempo e Estudos Climáticos). O objetivo principal é propor abordagens para reduzir os custos financeiros da execução do BRAMS em uma nuvem pública, especificamente na *Amazon Web Services* (AWS). A AWS oferece a ferramenta *ParallelCluster*, que facilita a criação de *clusters* de HPC em sua infraestrutura. Duas estratégias são investigadas: (i) o uso exclusivo de instâncias *Spot*, mais econômicas porém sujeitas a interrupção, e (ii) um algoritmo de seleção de instâncias que compara preços entre diferentes tipos, regiões e mercados. Este algoritmo recomenda a opção mais econômica e garante a continuidade da execução por meio da migração para instâncias *On-Demand*, que apesar de mais custosas, não estão sujeitas a revogação.

As principais contribuições são: (1) comparar o desempenho do BRAMS na nuvem e no supercomputador Santos Dumont, referência *On-Premises*; (2) propor mecanismos de tolerância a falhas para instâncias *Spot*; (3) desenvolver um algoritmo de seleção de instâncias baseado em custo e disponibilidade; e (4) avaliar estratégias de reescalonamento em instâncias *Spot* e *On-Demand*. Os resultados beneficiam o BRAMS e outras aplicações paralelas baseadas em MPI, ampliando o uso eficiente da nuvem para HPC.

2. Modelo regional BRAMS

O *Brazilian developments on the Regional Atmospheric Modeling System* (BRAMS) [Freitas et al. 2009] é um modelo numérico regional derivado do RAMS [Cotton et al. 2003], desenvolvido pelo INPE/CPTEC com funcionalidades que melhor representam os fenômenos meteorológicos tropicais. Computacionalmente, o BRAMS explora paralelismo via MPI, evoluindo de um modelo mestre-escravo para uma decomposição de domínio distribuída [Fazenda et al. 2011]. O BRAMS possui os modos de execução *initial*, onde a simulação é executada do início ao fim, e o modo *history*, no qual a simulação é executada a partir de um *checkpoint* [Walko et al. 2002]. Neste estudo foi utilizada a versão 5.2 [1], que dá suporte ao modo *history*.

3. Estratégia de reescalonamento em Spot

A estratégia proposta utiliza os mecanismos do *AWS ParallelCluster* com o escalonador *Slurm Workload Manager* para criar um ambiente tolerante a falhas, permitindo o uso de instâncias *Spot* para redução de custos. O *Slurm* gerencia o acesso aos recursos computacionais e oferece reescalonamento de tarefas em caso de falhas. Ao combinar o *AWS ParallelCluster* com instâncias *Spot* e o *Slurm*, em caso de revogação da instância, a execução é reiniciada em uma nova instância se for estática, ou restabelecida na mesma se for dinâmica. Para garantir a continuidade da execução após reinicializações causadas por revogações em instâncias *Spot*, foi desenvolvido um *script* de submissão que utiliza a variável de ambiente `SLURM_RESTART_COUNT` para identificar se uma execução é inicial ou uma reinicialização. Na primeira execução, o BRAMS é executado em modo *initial*. Em caso de reinicialização, um segundo *script* é acionado para configurar automaticamente o arquivo de configuração da execução do modelo (*namelist*) para o modo *history*, selecionando o *checkpoint* mais recente como ponto de reinício.

[1] Código-fonte disponível em: https://github.com/robertopsouto/BRAMS-5.2-modified/tree/fixbugs

Os experimentos compararam o desempenho do supercomputador Santos Dumont (2x Intel Xeon Gold 6252, 48 núcleos, 384 GB RAM) com um *cluster* na AWS utilizando instâncias `r5n.12xlarge` (48 vCPUs, 384 GB RAM). O modelo BRAMS foi configurado com uma grade de 180x180 pontos, 35 níveis verticais, previsão de 24 horas e *timesteps* de 30 segundos. Foram realizadas execuções sem gravação de *checkpoint* e com *checkpoints* a cada 1, 2 ou 3 horas de previsão para medir a sobrecarga desta operação, sendo que essas configurações, incluindo a frequência dos *checkpoints*, podem ser definidas no arquivo de configuração da previsão. A análise de desempenho revelou que o BRAMS apresentou um *speedup* significativamente maior na nuvem em comparação ao ambiente *On-Premises*. Enquanto no Santos Dumont o *speedup* com 4 nós foi de 3,67, na AWS alcançou 5,69 usando instâncias *Spot*. Esse ganho expressivo ocorreu porque o tempo de execução de referência (com um único nó) foi consideravelmente maior na nuvem: 4.059,1 segundos no Santos Dumont, contra 6.490,5 segundos usando instâncias *On-Demand* e 6.498,7 segundos em instâncias *Spot*. Conforme mais nós foram adicionados, a diferença entre os tempos de execução diminuiu: 1.330,7 segundos no Santos Dumont, 1.199,8 segundos na *On-Demand* e 1.141,4 segundos na *Spot* com quatro nós. Dessa forma, o *speedup* calculado para a nuvem tornou-se superior, embora isso não reflita em um desempenho melhor. O tempo de execução elevado na nuvem com um único nó pode ser atribuído a uma possível contenção no acesso à memória ou em operações de entrada/saída. Os resultados demonstraram que a utilização de instâncias *Spot* proporciona uma redução significativa de custos. Com 4 nós, o custo médio foi de $3,36 (Spot) contra $5,97 (On-Demand) para execuções sem falhas. A sobrecarga introduzida pela gravação de *checkpoints* mostrou-se baixa, aproximadamente 5 segundos por *checkpoint*, tanto em tempo de execução quanto em custo adicional (variação de centavos de dólar). Testes com injeção de revogações simuladas utilizando um script que encerra a instância seguindo a distribuição de Poisson[2], estratégia comumente empregada para simular eventos aleatórios no tempo, como revogações *Spot* [Duda 1983], demonstraram a eficácia da estratégia em finalizar a execução mesmo com múltiplas interrupções. No entanto, o tempo e o custo totais aumentaram proporcionalmente ao número de falhas, alcançando valores superiores aos obtidos utilizando apenas instâncias *On-Demand*.

4. Estratégia de reescalonamento em *On-Demand*

Para reduzir os custos elevados das execuções utilizando a estratégia de reescalonamento em instâncias *Spot*, foi projetado um algoritmo que seleciona a instância com o menor custo, comparando os custos de instâncias em diferentes regiões e mercados (*On-Demand* e *Spot*). Para cada instância, ele calcula o custo total considerando tempo de execução e, no caso de instâncias *Spot*, adiciona o tempo adicional de *checkpoint*. A cada passo, seleciona as instâncias com melhor custo-benefício, retornando ao final as três opções mais baratas. Aliado a esse algoritmo, foi pensado em uma estratégia de reescalonamento em instâncias *On-Demand* em caso de revogação de instâncias *Spot*.

Para evitar múltiplas interrupções durante a execução ao utilizar instâncias *Spot*, foi proposta uma alternativa que assegura que a aplicação seja reescalonada para uma fila com instâncias *On-Demand* caso ocorra uma interrupção. A proposta envolve a criação de um *cluster* com três filas: uma fila com instâncias *Spot*, outra fila com instâncias *On-Demand* (ambas do mesmo tipo de instância selecionada para a aplicação) e uma terceira

[2]Aplicação disponível em: https://github.com/alan-lira/vm-revoker

fila, chamada Controladora, para monitorar a execução. O fluxo de execução proposto envolve submeter um trabalho via *Slurm* para a fila Controladora, que o ressubmete para a fila *Spot* ou *On-Demand*. Se o trabalho for executado em instâncias *Spot*, sua execução é monitorada pela fila Controladora usando comandos Slurm. Se o estado do *job* for "*Completing*", indica uma interrupção e o trabalho é movido para a fila *On-Demand*. Se o estado não for "*Completing*", o estado do *job* é verificado novamente após um intervalo de tempo. O algoritmo é finalizado quando o valor de *job_status* retornado é *Not_Found*.

5. Resultados Experimentais

Os testes desta seção, realizados com base nos preços do primeiro semestre de 2024, avaliam o desempenho do BRAMS integrado à abordagem proposta de seleção e reescalonamento de instâncias. Um estudo comparativo foi realizado para avaliar o desempenho do modelo BRAMS em diversas instâncias AWS, abrangendo famílias otimizadas para computação, memória e HPC. O objetivo foi identificar instâncias com boa relação custo-benefício. Foram selecionadas instâncias com 48 vCPUs e de diferentes arquiteturas (x86: Intel/AMD; ARM: Graviton). Instâncias com GPU foram excluídas, pois o BRAMS não é projetado para arquitetura *many-core*. Foram consideradas as regiões *us-east-1*, *us-east-2*, *us-west-1*, *us-west-2* e *sa-east-1* nos mercados *Spot* e *On-Demand*. Foram analisados dois cenários: um com instâncias x86 e outro com instâncias x86 e ARM. As análises comparam custos e tempo de execução em um domínio menor, sem revogação e com revogação simulada de instâncias *Spot*, e em um domínio maior, considerando cenários reais de revogação.

Ao comparar os custos de instâncias *On-Demand* e *Spot*, a estratégia baseada no uso de instâncias *Spot*, mostrou-se vantajosa na ausência de revogações, com reduções de custo de até 89,80% (para a instância *c6gn.12xlarge* em *us-west-2*) em comparação com o mercado *On-Demand*. As maiores economias foram observadas no uso de instâncias ARM, que apresentaram preços *Spot* significativamente inferiores, variando entre $0,21 e $0,47 por hora de uso. Foram realizados testes de revogações de instâncias *Spot* utilizando a aplicação que simula os eventos de interrupção seguindo a distribuição de Poisson. A estratégia proposta consistiu em utilizar a ferramenta para selecionar e encerrar uma instância *Spot* atuando como nó de computação, reescalonando a execução para a fila *On-Demand* para evitar novas interrupções. Os experimentos foram conduzidos simulando revogações em três etapas distintas da execução do modelo: início (definido como até 25% do total de *timesteps*), meio (entre 25% e 75% do total de *timesteps*) e fim (após 75% do total de *timesteps*).

Os resultados demonstraram que o tempo total de execução, combinando o uso de instâncias *Spot* e *On-Demand*, sempre excedeu o tempo de uma execução realizada integralmente em *On-Demand*. Este acréscimo é atribuído ao tempo despendido com a escrita de *checkpoints* e com a reinicialização do processamento após uma revogação. Contudo, o impacto financeiro foi atenuado. Para a arquitetura x86, no cenário menos favorável (revogação no início), o tempo aumentou 55,34%, mas o custo total aumentou apenas 15,14%; já no cenário mais favorável (revogação no fim), registrou-se uma economia de custo de 12,53%. Com a inclusão de instâncias ARM, a economia de custo no melhor caso atingiu 15,36%. Uma análise mais aprofundada revelou que os arquivos relacionados ao cálculo da radiação, gerados durante o pré-processamento, estavam sendo recriados integralmente após cada reescalonamento para a fila *On-Demand*, impactando negativamente

o tempo de execução. Uma vez que estes arquivos precisam ser criados apenas uma vez, uma análise foi realizada descontando esse tempo de recriação. Esta otimização resultou em reduções significativas. Para a arquitetura x86, a economia de custo no cenário de revogação no fim saltou para 62,16%. Para a arquitetura ARM, a redução foi ainda mais expressiva, alcançando 72,78% no mesmo cenário. Em comparação com execuções totalmente em *On-Demand*, a estratégia otimizada permitiu economias de custo de até 66,93% (x86) e 76,92% (ARM). Estes resultados evidenciam o benefício da estratégia e a necessidade de implementar a reutilização dos arquivos de radiação.

Em um cenário mais próximo da realidade utilizando um domínio espacial maior no AWS ParallelCluster e no Santos Dumont, a configuração do modelo BRAMS utilizou uma grade de 680 x 740 pontos horizontais e 40 níveis verticais, com uma resolução espacial de 5 km. O tempo de previsão foi de 48 horas, utilizando *timesteps* de 10 segundos, totalizando 17.280 intervalos de integração. O intervalo de *checkpoint* foi configurado para três horas de previsão. Foram realizadas 10 execuções utilizando instâncias *Spot* de arquitetura ARM escolhidas pelo algoritmo de seleção. Não foram observadas revogações de instâncias *Spot* pela AWS. Os resultados demonstraram uma significativa economia de custo, atingindo uma redução de até 89,02% em comparação com o uso de instâncias *On-Demand*. Para fins de comparação, três execuções foram realizadas no supercomputador Santos Dumont, utilizando 4 nós, cada um com 2 processadores Intel Xeon Cascade Lake Gold 6252 (48 núcleos por nó). O tempo de execução no Santos Dumont foi ligeiramente maior que na nuvem. No entanto, o tempo de espera na fila do gerenciador de *jobs* Slurm no ambiente *on-premises* foi considerável, chegando a mais de 84 horas em um dos casos, resultando em um tempo de *turnaround* total muito superior ao observado na nuvem.

6. Conclusão

A migração de aplicações HPC para a nuvem requer considerações sobre desempenho e custo. Este estudo investigou a execução do modelo BRAMS em um *cluster* na AWS utilizando o *AWS ParallelCluster*. O trabalho propôs e avaliou estratégias para otimizar a execução na nuvem, com foco central na redução de custos. O uso de instâncias *Spot* mostrou-se uma ferramenta fundamental para diminuir custos, embora a possibilidade de revogações exija mecanismos robustos de contingência. Foram desenvolvidas três estratégias principais: 1) Uso de instâncias *Spot* com um *script* de reescalonamento de *jobs* via Slurm para tolerância a falhas; 2) Um algoritmo de seleção de instâncias que compara preços entre regiões e tipos (*Spot* e *On-Demand*); e 3) Um mecanismo de reescalonamento para instâncias *On-Demand* em caso de revogação. Os resultados foram significativos: o algoritmo de seleção alcançou reduções de custo de até 89,80% em comparação com instâncias *On-Demand*. Instâncias com arquitetura ARM consistentemente apresentaram custos mais baixos que as x86. O reescalonamento para instâncias *On-Demand* mostrou-se vantajoso financeiramente quando as revogações ocorriam no meio ou no final da execução, pois a maior parte do processamento era realizada em instâncias *Spot*.

A comparação com o ambiente *on-premises* revelou que o tempo de execução na nuvem foi ligeiramente menor, possivelmente devido a hardware mais moderno. Contudo, a vantagem mais decisiva residiu no tempo de *turnaround*, drasticamente reduzido na nuvem, uma vez que o ambiente local sofre com longos tempos de espera na fila do Slurm, que podem durar dias, devido à alta demanda sazonal pelos recursos do supercomputador. A combinação do *cluster* na nuvem com as estratégias propostas

mostrou-se promissora para a execução do BRAMS, equilibrando custo e eficiência temporal. Para trabalhos futuros, destacam-se: monitorar a variação de preços das instâncias *Spot*, desenvolver uma estratégia para reutilizar arquivos de cálculo de radiação, investigar as causas dos *speedups* elevados observados e testar a execução do modelo em um *cluster* na nuvem com mais nós. Os resultados deste trabalho foram divulgados por meio de uma publicação no Simpósio em Sistemas Computacionais de Alto Desempenho [Melo et al. 2023] e um artigo no periódico *Concurrency and Computation: Practice and Experience* [de Melo et al. 2025]. Além disso, os resultados foram apresentados ao Comitê Científico do *Model for Ocean-laNd-Atmosphere predictioN* (MONAN), um programa institucional do INPE e Ministério da Ciência e Tecnologia (MCTI) para desenvolvimento do novo modelo comunitário brasileiro do sistema terrestre.

Referências

Cotton, W. R., Pielke Sr, R., Walko, R., Liston, G., Tremback, C., Jiang, H., McAnelly, R., Harrington, J., Nicholls, M., Carrio, G., et al. (2003). Rams 2001: Current status and future directions. *Meteorology and Atmospheric Physics*, 82(1-4):5–29.

de Melo, M. S., Souto, R. P., and Drummond, L. M. (2025). Optimized execution of a numerical weather forecast model in a cloud cluster. *Concurrency and Computation: Practice and Experience*, 37(3):e8374.

Duda, A. (1983). The effects of checkpointing on program execution time. *Information Processing Letters*, 16(5):221–229.

Fazenda, A. L., Panetta, J., Katsurayama, D. M., Rodrigues, L. F., Motta, L. F., and Navaux, P. O. (2011). Challenges and solutions to improve the scalability of an operational regional meteorological forecasting model. *International Journal of High Performance Systems Architecture*, 3(2-3):87–97.

Freitas, S. R., Longo, K. M., Silva Dias, M. A. F., Chatfield, R., Silva Dias, P., Artaxo, P., Andreae, M. O., Grell, G., Rodrigues, L. F., Fazenda, A., and Panetta, J. (2009). The coupled aerosol and tracer transport model to the brazilian developments on the regional atmospheric modeling system (catt-brams)–part 1: Model description and evaluation. *Atmospheric Chemistry and Physics*, 9(8):2843–2861.

Melo, M. S., Drummond, L. M., and Souto, R. P. (2023). In portuguese: Análise de custo e desempenho de um sistema de modelagem atmosférica tolerante a falhas no aws parallelcluster. pages 217–228. Proceedings of XXIV Symposium on High Performance Computing Systems (WSCAD), SBC.

Michalakes, J. (2020). Hpc for weather forecasting. *Parallel Algorithms in Computational Science and Engineering*, pages 297–323.

Powers, J. G., Werner, K. K., Gill, D. O., Lin, Y.-L., and Schumacher, R. S. (2021). Cloud computing efforts for the weather research and forecasting model. *Bulletin of the American Meteorological Society*, 102(6):E1261–E1274.

Walko, R. L., Tremback, C. J., Panetta, J., Freitas, S., and Fazenda, A. L. (2002). *RAMS - Regional Atmospheric Modeling System Version 5.0: Model input namelist parameters*. CPTEC.