Análise de Custo e Desempenho de um Sistema de Modelagem Atmosférica Tolerante a Falhas no *AWS ParallelCluster*

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 aimed to analyze the performance of BRAMS numerical weather prediction model running on an AWS cluster created with AWS ParallelCluster in different instance markets and compare it when running on Santos Dumont Supercomputer. A methodology was proposed to run a fault-tolerant BRAMS version in the spot market, where the instances can be revoked, although they offer lower costs. The obtained execution times in clouds were satisfactory when compared to Santos Dumont. In general, the spot solution reduced the financial cost when compared to the use of regular On-Demand instances. Only in a scenario with many revocations, which consequently increases the execution time and the cost, the option of using the On-Demand market was more suitable.*

Resumo. *Este trabalho teve como objetivo analisar o desempenho do modelo de previsão numérica do tempo BRAMS em execução em um cluster AWS criado com o AWS ParallelCluster em diferentes mercados de instâncias, comparando-o com a execução no supercomputador Santos Dumont. Foi proposta uma metodologia para executar uma versão tolerante a falhas do BRAMS no mercado de Spot, onde as instâncias podem ser revogadas, embora ofereçam custos mais baixos. Os tempos de execução na nuvem foram satisfatórios quando comparados ao Santos Dumont. Em geral, a solução Spot reduziu o custo financeiro quando comparado ao uso de instâncias regulares On-Demand. Apenas em um cenário com muitas revogações, o que consequentemente aumenta o tempo de execução e o custo, a opção de usar o mercado On-Demand foi mais adequada.*

1. Introdução

Os provedores de nuvem oferecem provisionamento de recursos sob demanda, com alta disponibilidade e escalabilidade, em um sistema de pagamento conforme a utilização dos serviços. Os modelos mais comuns de serviços são Infraestrutura como Serviço, também conhecido como *Infrastructure-as-a-Service* (IaaS), Plataforma como Serviço ou *Platform-as-a-Service* (PaaS) e *Software* como Serviço ou *Software-as-a-Service* (SaaS). Mas também podem ser oferecidos modelos mais especializados, como Computação de Alto Desempenho como Serviço, High Performance Computing as a Service (HPCaaS) [Bourhnane and Abid 2020] e Função como Serviço, Function as a Service (FaaS) [Castro et al. 2017], entre outros.

O interesse em adotar a computação em nuvem como uma plataforma alternativa para executar aplicações de HPC (High-Performance Computing) vem crescendo, ultimamente. Alguns estudos iniciais sobre a avaliação do uso da computação em nuvem para aplicações de HPC, no entanto, expuseram algumas de suas limitações [He et al. 2010] [Netto et al. 2018]. Aplicações de HPC fortemente acopladas, que dependem de comunicação frequente entre processos, podem não apresentar bom desempenho e escalabilidade em ambientes de nuvem, devido ao baixo desempenho da rede e o compartilhamento dos processadores. A maioria das aplicações de HPC é executada em múltiplos nós de grande poder computacional e largura de banda de rede dedicada. Esse ambiente de execução não é tradicionalmente oferecido pela nuvem. Para preencher a lacuna entre a computação em nuvem e o HPC, vários provedores buscaram oferecer serviços voltados computação de alto desempenho. A AWS oferece um serviço que permite a criação de clusters para HPC, chamado ParallelCluster [Amazon Web Services 2023b]. O ParallelCluster é uma ferramenta de código aberto, criada em 2018, que permite montar e gerenciar *clusters* de alto desempenho. Através de um arquivo de configuração, é possível definir as características do *cluster* paralelo, que incluem (a) o número de nós e a(s) instância(s) que comporão cada nó; (b) a rede; (c) o tipo e tamanho do armazenamento; e (d) outros serviços da AWS. Com um *cluster* configurado, a AWS permite ao usuário enviar tarefas (jobs), suportando *job schedulers* como o AWS Batch e o Slurm. Não há custo para o uso da ferramenta AWS ParallelCluster. No entanto, a AWS cobra pelas instâncias, rede, armazenamento e outros serviços incluídos em um *cluster* específico.

A AWS oferece três principais modelos de mercado de pagamento de instâncias: *On-Demand*, *Spot* e *Reserved* [Amazon Web Service 2023]. Nos modelos *On-Demand* e *Spot* o pagamento ocorre conforme o uso (ou seja, cobrados por hora/segundo de uso), enquanto no *Reserved* o usuário pode contratar as instância por períodos que podem variar entre 1 ou 3 anos. No modelo *On-Demand*, cada instância tem um custo fixo por hora/segundo, que será cobrado desde o momento em que a instância é adquirida até o momento em que o usuário a encerra. Por outro lado, as instâncias *Spot* têm um custo variável por hora/segundo e geralmente são muito mais baratas do que as correspondentes *On-Demand*. No entanto, as instâncias *Spot* podem ser revogadas pelo provedor a qualquer momento e, por esse motivo, as aplicações que são executadas em instâncias *Spot* devem estar preparadas para essa situação com um mecanismo de tolerância a falhas.

No campo da meteorologia, encontra-se diversos modelos numéricos de previsão de tempo e clima, que se beneficiam com o uso da computação de alto desempenho [Michalakes 2020], pois há uma grande quantidade de dados a serem processados e equações a serem resolvidas, para atualizar o avanço do estado da atmosfera. O uso de nuvens para computação de alto desempenho vem sendo explorada nessa área, com estudos que analisam a viabilidade de utilização com vários modelos [Montes et al. 2020] [Powers et al. 2021] [Koop and Raman 2021]. Este trabalho será focado no BRAMS (*Brazilian developments on the Regional Atmospheric Modeling System*), um modelo numérico de previsão do tempo e clima de escala regional mantido pelo INPE/CPTEC (Instituto Nacional de Pesquisas Espaciais/Centro de Previsão de Tempo e Estudos Climáticos), baseado no modelo RAMS (*Regional Atmospheric Modeling System*) [Pielke et al. 1992]. O objetivo principal desta pesquisa é analisar os tempos e custos financeiros de execução do BRAMS no *ParallelCluster*, usando instâncias *On-demand* e

Spot. Neste último caso, consideramos os custos de gravar *checkpoints* e recuperar a aplicação em caso de falhas em diversos cenários. Ainda, realizamos uma comparação de tempo de execução, considerando o supercomputador (*On-Premise*) Santos Dumont.

O restante deste trabalho está organizado da seguinte forma. Na Seção 2, há uma descrição do modelo BRAMS e alguns aspectos do seu funcionamento que foram necessários para a realização desse trabalho. Alguns trabalhos que estudaram a execução do BRAMS na nuvem são citados na Seção 3. Já na Seção 4, são descritas as metodologias propostas para a execução do modelo em nuvem de modo que fosse tolerante a falhas. Os resultados dos testes realizados são descritos na Seção 5. Finalmente, a Seção 6 conclui o trabalho e apresenta passos futuros.

2. Modelo regional BRAMS

O modelo regional *Brazilian developments on the Regional Atmospheric Modeling System* (BRAMS) [Freitas et al. 2017] foi desenvolvido principalmente pelo INPE/CPTEC, baseando-se no modelo *Regional Atmospheric Modeling System* (RAMS) [Pielke et al. 1992], lançado na década de 1980 pela *Colorado State University* nos Estados Unidos. O BRAMS oferece um modelo especializado na previsão regional de tempo e de clima no Brasil [Freitas et al. 2016], diferenciando-se do RAMS sobretudo ao introduzir novas funcionalidades que melhor representam os fenômenos meteorológicos tropicais [Freitas et al. 2009]. Do ponto de vista computacional, foram implementadas alterações que visaram o paralelismo massivo do modelo. Como estratégia de paralelização, foi empregada a decomposição do domínio da previsão, onde a atualização da atmosfera é realizada em paralelo nas diferentes sub-regiões do domínio. A implementação desta estratégia foi realizada por meio da biblioteca de passagem de mensagens MPI, permitindo escalabilidade de 9600 núcleos computacionais [Fazenda et al. 2012] de acordo com os critérios operacionais do INPE/CPTEC, sendo possível rodar uma previsão de 5Km em 20 minutos de tempo de execução.

O BRAMS possui diferentes tipos de execuções [Walko et al. 2002]. Entre eles, pode-se destacar a execução contínua, chamada na aplicação de *initial*, onde a simulação é feita do início ao fim. Há também o modo *history*, no qual a simulação é executada a partir de um *checkpoint*. O modo *history* é oportuno para a execução do modelo em um ambiente onde pode ocorrer falhas e está disponível na versão 5.2 [1], que foi utilizada neste estudo.

3. Trabalhos Relacionados

O uso de nuvens computacionais públicas na execução de modelos numéricos de previsão de tempo e clima tem ganhado interesse nos últimos anos, surgindo estudos que verificam a viabilidade de migração para nuvem, bem como seus desafios e uso de diferentes ferramentas para execução em nuvens computacionais. A respeito de sistemas de computação de alto desempenho tolerantes a falhas em nuvens computacionais, diversos estudos apontam a necessidade desses sistemas possuirem essa característica [Guedes et al. 2020][Sousa et al. 2023]. No campo da meteorologia, pode-se encontrar alguns trabalhos que também abordam esse tema [Xu et al. 2019]

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

[Benacchio et al. 2021], mas ainda há poucos estudos que avaliam o comportamento tolerante a falhas do BRAMS em nuvens computacionais. Entretanto, o seu uso em nuvem tem sido discutido por alguns autores.

Em [Carreno et al. 2015], os autores discutiram os desafios da migração e o desempenho do modelo BRAMS utilizando a nuvem Microsoft Azure como IaaS. Foi visto que o processamento e a rede são fatores limitante ao utilizar a nuvem, porém utilizá-la para o armazenamento dos dados se mostra uma opção pertinente para compartilhar resultados e facilitar a implantação de um banco de testes para uma plataforma de pesquisa meteorológica. Além disso, foi concluído que a utilização da nuvem é uma boa alternativa contanto que os provedores continuem atualizando seus ambientes e continuem baixando os preços de seus serviços. Já no trabalho [Carreno et al. 2016], os autores modificaram diversos procedimentos no pré-processamento e pós-processamento a fim de reduzir tarefas repetitivas. Utilizou-se a estratégia de armazenamento compartilhado para reduzir o tempo em casos em que várias instalações do BRAMS são usadas. Para os testes, foi executado o modelo em diversos nós computacionais. Há trabalhos que buscam diferentes métodos de execução em nuvem, como em [de Araujo et al. 2020], que buscou analisar o modelo BRAMS sendo executado puramente em nuvem e também utilizando contêineres em nuvem. Foi visto que para um caso pequeno, há um baixo *overhead* quando é utilizado um contêiner para cada processo e quando utilizados mais processos para cada contêiner, o tempo de execução fica semelhante a execução puramente na nuvem.

Com os trabalhos citados mostrando que é possível a execução do BRAMS em nuvem, apesar dos desafios, juntamente com a busca de diferentes técnicas para o aprimoramento da execução, o presente trabalho busca utilizar o modelo em um ambiente em nuvem especializado em HPC com técnicas conhecidas para otimização de custo, como a utilização de instâncias *Spot*.

4. Metodologia Proposta para Execução em Nuvem

A estratégia proposta utiliza a *Amazon Web Services* (AWS) como nuvem computacional pública. Para a criação do *cluster* em nuvem, foi escolhida a ferramenta *AWS ParallelCluster* com o escalonador *Slurm Workload Manager*. *AWS ParallelCluster* é uma ferramenta de código-aberto da Amazon que simplifica a criação e o gerenciamento de *cluster* de alto desempenho utilizando os recursos da própria AWS. Com essa ferramenta, é possível configurar facilmente as instâncias que serão utilizadas, incluindo o tipo de mercado *On-Demand* ou *Spot*, o sistema de arquivo desejado e também o escalonador. Não há custo ao utilizar o *AWS ParallelCluster*, porém são cobrados os recursos utilizados no *cluster* criado por ele. *Slurm Workload Manager* [Yoo et al. 2003], ou mais conhecido como *Slurm*, é um escalonador de tarefas de código-aberto muito utilizado em *clusters* de diversos tamanhos. Sua função é gerenciar o acesso dos usuários aos recursos computacionais de maneira eficiente, garantindo alta escalabilidade e reexecução de tarefas em caso de falhas. Também é possível gerenciar o ciclo de vida de uma tarefa, como iniciar, executar e monitorar o seu estado.

Pode-se aproveitar do recurso de reescalonamento de tarefas do *Slurm* para diminuir o custo de execução na AWS, utilizando instâncias *Spot* na execução de aplicações tolerantes a falhas. Ao configurar o *AWS ParallelCluster* com a combinação de *Slurm* com instâncias *Spots*, em caso de falhas enquanto a aplicação está sendo executada ou a

AWS requisite o recurso utilizado pela instância *Spot*, a execução é reiniciada em uma nova instância caso ela seja estática, ou caso seja dinâmica, a execução é restabelecida quando a instância for reiniciada.

4.1. Execução do BRAMS com tolerância a falhas no *AWS ParallelCluster*

O BRAMS em modo *history*, que permite a execução a partir de um *checkpoint*, fornece um modelo de *checkpoint* sincronizado, onde o processo principal reúne as informações da decomposição de domínio fornecida pelos demais processos e então constrói os arquivos de saída. Para poder executar o modelo BRAMS nesse modo, é necessário primeiro ter rodado a execução contínua com as variáveis `IOUTPUT`, `HFILOUT`, `FRQHIS` configuradas no arquivo de *namelist* (*RAMSIN*). Essas variáveis são responsáveis respectivamente pelo formato do arquivo de *checkpoint*, o local que os arquivos serão salvos, bem como o prefixo dos arquivos, e também a frequência em tempo de previsão que os *checkpoints* serão salvos. Após essa execução, basta configurar no arquivo de *namelist* as opções `TIMSTR` e `HFILIN`, onde é possível configurar o tempo que a previsão irá ser iniciada no modo *history* e também o arquivo de *checkpoint* do tempo de início.

A possibilidade de reescalonamento de instâncias e execuções que o *AWS ParallelCluster* e o *Slurm* trazem, e a capacidade do modelo BRAMS de iniciar uma execução a partir de um *checkpoint* são imprescindíveis para a execução em um *cluster* criado com instâncias *Spot* a fim de obter custos menores. Para a utilização desses três recursos (*AWS ParallelCluster*, *Slurm*, e *Checkpoints*) de forma automática, sem a interação do usuário, foi desenvolvido um procedimento que provê ao usuário a utilização da nuvem nesse ambiente de forma transparente. Sem o desenvolvimento do procedimento proposto, o *job* ao ser reiniciado iria sempre executar o modelo em modo *initial*, começando a previsão do início. Além disso, para executar o modelo em modo *history* seria necessário configurar algumas variáveis manualmente, como explicado anteriormente. Para resolver essas questões, foi criado um *script* [2] que automatizasse todo processo.

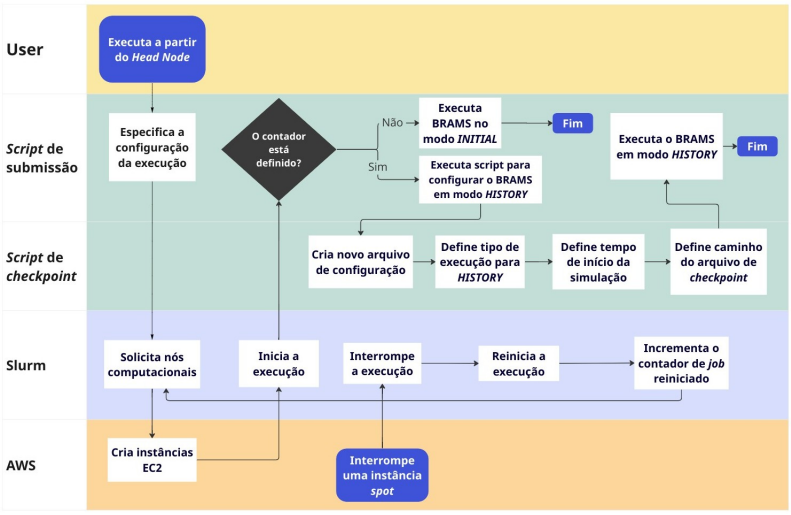

Figura 1. Fluxograma da execução do BRAMS em um ambiente com falhas

A Figura 1 apresenta o fluxo das ações que ocorre ao utilizar o *script* de automação

[2]https://github.com/robertopsouto/BRAMS-5.2-modified/tree/fixbugs/scripts

em um cluster criado com *AWS ParallelCluster*, utilizando instâncias *Spots* e o *Slurm* como escalonador. Ao submeter um *job*, o *script* de submissão especifica as configurações da execução, como o número de processos e quantos nós são necessários. Após isso, são criadas as instâncias configuradas no passo anterior e a execução começa. Logo no início, há uma condicional no *script* que serve para verificar se é a primeira execução ou se já houve falhas. Esse controle de execução é definido pela variável de ambiente do *Slurm*, `SLURM_RESTART_COUNT`. Se ela não estiver declarada, significa que é a primeira vez que o *job* está sendo executado. Caso contrário, o seu valor será a *n-ésima* reexecução do *job*. Com essa informação, é possível configurar o modelo BRAMS para ser executado em modo *initial*, caso seja a primeira execução, ou em modo *history*, caso não seja.

Seguindo o fluxo da primeira execução, a aplicação será apenas executada normalmente, já que o *script* de submissão recebe como argumento o caminho do arquivo de *namelist* para ser executado em modo *initial*. Em caso de reexecução, é chamado um segundo *script*, responsável por automatizar as configurações do modo *history*. Esse *script* cria um novo arquivo *namelist* a partir do utilizado na fase *initial*, mas substituindo o tipo de execução para *history*. Em seguida, são modificadas as variáveis responsáveis pelo tempo de início e pelo arquivo de *checkpoint* a ser utilizado. Essas informações são obtidas no próprio arquivo de *namelist*, através da variável que guarda o caminho que os *checkpoints* foram salvos. Então, é selecionado o arquivo mais recente para preencher as informações. Após essas configurações, a execução volta para o script de submissão rodando o modelo a partir do arquivo recém-criado.

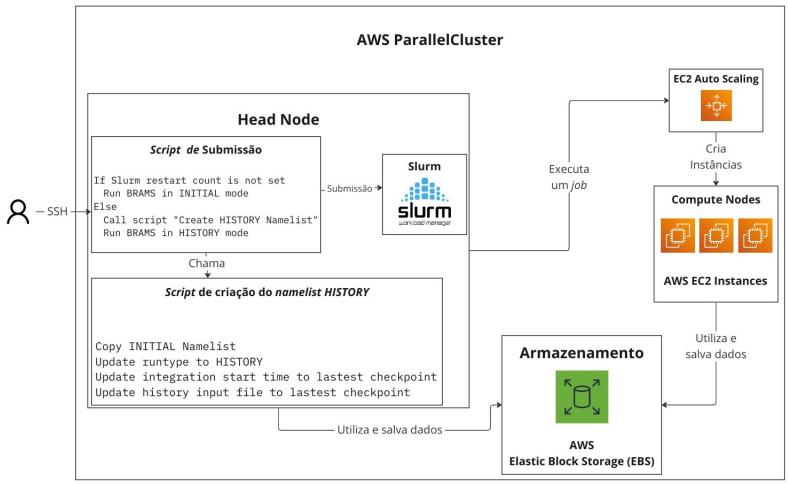

Figura 2. Arquitetura da execução do BRAMS no *AWS ParallelCluster*

Do ponto de vista da infraestrutura, a Figura 2 mostra a execução do BRAMS no *AWS ParallelCluster*, desde a submissão em um *head node*, ao qual o usuário se conecta via *ssh*. Esse tipo de instância utiliza o mercado *On-Demand*. A partir do *head node*, é possível utilizar o *Slurm* para submeter os *jobs* usando o script de submissão, o mesmo que é responsável por definir o melhor tipo de execução, como explicado anteriormente. Ao submeter um job, é utilizado o serviço *EC2 Auto Scaling* para instanciar os nós de execuções (*compute nodes*). Esses nós podem ser do tipo *On-Demand* ou *Spot*, de acordo com que for configurado na criação do cluster. As instâncias têm acesso ao mesmo local de armazenamento, o AWS Elastic Block Storage (*EBS*), na nossa proposta.

O serviço *AWS Fault Injection Simulator* (*AWS FIS*) [Amazon Web Services 2023a] foi utilizado para testar o funcionamento da estratégia proposta durante o seu desenvolvimento. Esse serviço serve para injetar falhas em recursos utilizados na AWS, para analisar o comportamento da aplicação em condições adversas no ambiente de produção, avaliando a sua robustez em um ambiente com interrupções. Para executar experimentos com o AWS FIS, é necessário criar um modelo de experimento primeiro. Esse modelo é a base do experimento e contém as ações, alvos e condições de parada do teste. Após criar o modelo de experimento, é possível executá-lo e assim acompanhar o progresso e visualizar o estado dos testes em execução.

4.2. Ambientes de Execução Santos Dumont e *AWS ParallelCluster*

Como referência de execução da aplicação em ambiente *On-Premise*, utilizamos o supercomputador Santos Dumont. Foram alocados de forma exclusiva os nós computacionais que continham dois processadores *Intel Xeon Cascade Lake Gold 6252* com 24 núcleos cada, totalizando 48 núcleos, e com 384GB de memória principal. O sistema operacional utilizado foi o RedHat Linux 7.6. Para a execução no AWS ParallelCluster, foi escolhido montar um *cluster* com instâncias *r5n.12xlarge* tanto no *head node*, nó que o usuário faz o *login* e gerencia os nós computacionais, quanto nos *compute nodes*, nós que servem para o processamento da aplicação. Essa instância possui processadores Intel Xeon Platinum 8259, 48 vCPUs e 384GB de memória principal. Embora esse tipo de instância conte com o modelo de processador diferente dos alocados no Santos Dumont, essa instância foi a que se aproximou melhor das configurações do supercomputador, levando em consideração a capacidade da memória principal, o número de núcleos e vCPUs equivalentes e a geração de processadores *Cascade Lake*. Os testes na nuvem foram separados em dois grupos distintos, um grupo utilizando instâncias *On-Demand* nos *compute nodes*, onde a máquina virtual é reservada especialmente para o usuário, e outro grupo usando instâncias *Spot* nos *compute nodes*, que são máquinas virtuais que usam a capacidade sobressalente disponível por um menor preço, mas que podem ser interrompidas a qualquer momento.

5. Resultados Experimentais

Os testes foram realizados com a previsão configurada com a resolução espacial de 10Km x 10Km e com a grade de 180 x 180 pontos, com 35 níveis verticais. O tempo de previsão foi de 24 horas com *timestep* de 30 segundos, totalizando 2880 *timesteps*, e com o cálculo do esquema da radiação na atmosfera sendo feito a cada 120 segundos, totalizando 720 *timesteps* com a radiação ativada. O modelo foi executado nas máquinas utilizando 1, 2 e 4 nós computacionais com 48 processos MPI em cada nó. Foram feitas até cinco execuções para cada teste, comparando a média do tempo total de execução e o custo médio da execução, nos casos de teste na AWS.

A Tabela 1 apresenta o resultado das execuções comparando o tempo de execução em segundos em cada sistema e o seu ganho de desempenho (*speedup*). Percebe-se que nas execuções com apenas 1 nó, o tempo de execução no Santos Dumont foi bem menor comparado às execuções na AWS, mas essa diferença diminuiu à medida que se inseriu mais nós computacionais. A aplicação teve um bom ganho de desempenho ao acrescentar mais nós, com destaque para as execuções na AWS, que tiveram um *speedup* acima do esperado, tendo a execução com 4 nós utilizando o mercado *Spot* alcançado 5,69 de

speedup. Uma hipótese sobre o ganho expressivo de desempenho é o tamanho da granularidade dos dados quando é executado em apenas 1 nó, podendo ocorrer contenção no acesso à memória ou na entrada/saída.

Tabela 1. Comparação dos tempos de execução, custo e *Speedup* no Santos Dumont, AWS *On-Demand* e *Spot*.

	Tempo (s)			*Speedup*			Custo Total	
# Nós	SDumont	*On Demand*	*Spot*	SDumont	*On Demand*	*Spot*	*On Demand*	*Spot*
1	4.059,1	6.490,5	6.498,7	1,00	1,00	1,00	$13,31	$10,02
2	1.943,2	2.288,2	2.299,1	2,09	2,84	2,83	$7,18	$4,87
4	1.330,7	1.199,8	1.141,4	3,05	5,41	5,69	$6,30	$3,69

Além do tempo de execução e *speedup*, foram analisados os custos para utilizar o BRAMS na AWS. Foi cobrado $3,576 por hora de uso das instâncias *On-Demand* e em média $1,7441 por hora de uso das instâncias *Spot*. A Tabela 1 faz um comparativo do custo de cada execução na AWS. Os resultados foram calculados somando o quanto foi cobrado pela utilização do *head node* e dos *compute nodes*. Além disso, foi verificado através do *Console* de Faturamento da AWS que foi cobrado $0,045 por hora de serviço *EC2 NatGateway*, $0,29 pelo serviço *Elastic File System* e $0,04 pelo serviço *Route S3*.

O próximo passo foi verificar a sobrecarga do tempo de execução ao utilizar a opção de fazer *checkpoint*. Foi feita a comparação dos tempos sem *checkpoint*, *checkpoint* sendo feito a cada 1 hora de previsão, totalizando 24 *checkpoints*, sendo feito a cada 2 horas de previsão, totalizando 12 *checkpoints*, e sendo feito a cada 3 horas de previsão, totalizando 8 *checkpoints*. A Tabela 2 apresenta os resultados dessa comparação, onde é possível perceber que nos três sistemas há a sobrecarga no tempo de execução quando utilizada essa funcionalidade. Mas, a sobrecarga diminui pouco ao escolher fazer menos *checkpoints*. A sobrecarga é maior no Santos Dumont ao utilizar mais nós computacionais, quase 200 segundos a mais na execução de *checkpoint* a cada 1 hora de previsão em 4 nós computacionais comparado a execução sem *checkpoint* com a mesma quantidade de nós. Esse comportamento pode ser sido causado pela limitação da rede para o sistema de arquivos *Lustre*. Além disso, as operações de entrada e saída podem ter sido afetadas pelas bibliotecas MPI instaladas no Santos Dumont, pois algumas implementações não alcançam bom desempenho nessas operações no supercomputador [Bez et al. 2020].

Tabela 2. Comparação dos tempos de execução sem e com gravação de *checkpoint* no Santos Dumont, AWS *On-Demand* e *Spot*.

	Sem Checkpoint			24 Checkpoints (1h)			12 Checkpoints (2h)			8 Checkpoints (3h)		
# Nós	SDumont	*On-Demand*	*Spot*	SDumont	*On-Demand*	*Spot*	SDumont	*On-Demand*	*Spot*	SDumont	*On-Demand*	*Spot*
1	4.059,1	6.490,5	6.498,7	4.192,3	6.568,0	6.606,4	4.128,2	6.550,3	6.518,6	4.118,9	6.531,6	6.544,8
2	1.943,2	2.288,2	2.299,1	2.109,8	2.380,0	2.464,3	2.012,5	2.351,7	2.375,8	1.997,4	2.344,5	2.356,3
4	1.330,7	1.199,8	1.141,4	1.514,6	1.246,7	1.234,5	1.404,2	1.212,7	1.203,7	1.367,8	1.203,3	1.197,3

Também foram comparados os custos das execuções sem e com *checkpoints*, como visto na Tabela 3. Percebe-se que o custo da sobrecarga é baixo, não ultrapassando alguns centavos de dólar. A maior diferença no custo entre as execuções que não foi gravado *checkpoint* e uma das opções de gravação é no cenário que foi utilizado o mercado *On-Demand* com *checkpoint* a cada 1 hora e utilizando 2 nós e também no cenário onde foi utilizado o mercado *Spot* com *checkpoint* a cada 1 hora e utilizando 2 nós, custando $0,27 e $0,33 respectivamente.

Tabela 3. Custo de utilizar AWS ParallelCluster nas execuções com e sem *checkpoint* em um nó computacional na AWS *On-Demand* e *Spot*.

	Sem checkpoint		24 Checkpoints (1h)		12 Checkpoints (2h)		8 Checkpoints (3h)	
# Nós	*On-Demand*	*Spot*	*On-Demand*	Spot	*On-Demand*	*Spot*	On-Demand	*Spot*
1	$13,31	$10,02	$13,46	$10,18	$13,43	$10,09	$13,39	$10,09
2	$7,18	$4,87	$7,45	$5,20	$7,37	$5,03	$7,35	$4,98
4	$6,30	$3,69	$6,54	$3,96	$6,37	$3,92	$6,32	$3,87

Verificou-se também o desempenho da rede entre os nós nos sistemas. Os nós no Santos Dumont são interligados com uma rede *Infiniband FDR* de 56Gb/s e a intância escolhida no *AWS ParallelCluster* possui uma largura de banda de 50Gb/s. Foi feito um teste enviando pacotes de 64 *bytes* para os nós e verificado o tempo de resposta. O tempo de resposta no Santos Dumont entre o nó principal e os nós computacionais foi de 0,119ms e entre os nós computacionais foi de 0,115ms. Já no *AWS ParallelCluster* com instâncias *Spot*, o tempo de resposta entre o *Head Node* e os *Compute Nodes* foi de 0,120ms e entre os *Compute Nodes*, 0,075ms. Já nas instâncias *OnDemand*, o tempo de resposta entre o *Head Node* e os *Compute Nodes* foi de 0,134ms e entre os *Compute Nodes*, 0,130ms

Tabela 4. Tempo e custo de utilizar um *cluster* com 2 nós criado a partir do AWS ParallelCluster em cenários com falhas.

	# Falhas				Custo		
Execução	Nó 1	Nó 2	Total	Tempo	*Head Node*	*Compute Nodes*	Total
1°	1	2	3	4.012	$3,99	$3,89	$8,25
2°	1	0	1	3.207	$3,19	$3,11	$6,66
3°	4	1	5	6.493	$6,45	$6,29	$13,15

Para testar a solução da execução em um cenário com falhas na nuvem, foi utilizado o *AWS ParallelCluster* configurado com 2 e 4 nós computacionais e com o *checkpoint* sendo gravado a cada 1 hora de previsão. Para avaliar os testes com diferentes padrões de falha, utilizou-se uma aplicação [3] que simula interrupções nas instâncias seguindo a distribuição de Poisson [Ahrens and Dieter 1974]. A distribuição de Poisson foi configurada com $\lambda_r = \frac{1}{1800}$ para 2 nós e $\lambda_r = \frac{1}{900}$ para 4 nós, que significa o intervalo médio de tempo em segundos entre a chegada dos eventos de revogação. Quando o evento de revogação ocorre, a instância corrente é encerrada e outra instância *Spot* é alocada para prosseguir com a execução da aplicação. A Tabela 4 e a Tabela 5 mostram os resultados nos testes da simulação de falhas com 2 e 4 nós respectivamente. Foram feitas 3 execuções, que tiveram diferentes quantidades de falhas. O custo da execução foi avaliado seguindo o mesmo cálculo anterior. Percebe-se que as falhas podem aumentar consideravelmente o tempo de execução e consequentemente o custo final, mesmo em casos de apenas uma falha. Isso ocorre devido ao modelo ter que executar a fase inicial, preparando os dados para a previsão, mesmo reiniciando uma execução a partir de um *checkpoint*. No pior cenário, em que há 9 falhas utilizando 4 nós, o custo total é $23,13, maior que o custo de utilizar instâncias *On-Demand*, $6,54.

Foram analisados arquivos de saída dos campos relativos a temperatura (`tempk`) e umidade relativa (`rh`), após 24 horas de previsão, obtidos com a execução contínua do modelo BRAMS e com as execuções com *checkpoint-restart*. Observaram-se algumas pequenas diferenças numéricas na execução com *checkpoint-restart*, que no entanto não comprometeram a qualidade do resultado de previsão destes dois campos.

[3] https://github.com/alan-lira/vm-revoker

Tabela 5. Tempo e custo de utilizar um *cluster* com 4 nós criado a partir do AWS ParallelCluster em cenários com falhas.

	# Falhas						Custo		
Execução	Nó 1	Nó 2	Nó 3	Nó 4	Total	Tempo	*Head Node*	*Compute Nodes*	Total
1°	1	2	2	4	9	7.744	$7,69	$15,01	$23,13
2°	0	1	1	0	2	3.069	$3,05	$5,95	$9,36
3°	1	0	0	0	1	2.233	$2,22	$4,33	$6,90

6. Conclusão

Pesquisas voltadas para a execução de modelos numéricos de previsão de tempo e clima em nuvens computacionais tem ganhado seu espaço, com trabalhos que analisam o seu desempenho, além de propostas de migração de execução em sistemas *on premise* para nuvem, e também propostas de utilização de outras ferramentas em conjunto, como contêineres. O presente trabalho tratou de analisar o desempenho e custo de execução do modelo BRAMS em nuvens computacionais utilizando o *AWS ParallelCluster*, com os mercados *On-Demand* e *Spot*. Também foi apresentada uma comparação com um sistema *on-premise*, para o qual foi escolhido o supercomputador Santos Dumont. Como a utilização de instâncias *Spot* pode acarretar em falhas durante a execução, para tolerar tais eventuais falhas, foi utilizado o mecanismo de ressubmissão de *jobs* do *Slurm*, escalonador escolhido na criação do *cluster*, juntamente com um *script* que verifica quando a aplicação BRAMS está sendo ressubmetida para escolher o seu melhor modo de execução. Embora o desempenho na nuvem tenha sido satisfatório, o seu custo de utilização continua sendo um fator negativo. Neste sentido, a opção de utilizar o mercado *Spot* torna a sua utilização mais atraente. A implementação proposta se mostrou eficiente ao simular diferentes quantidades de falhas durante a execução do modelo BRAMS. Porém em casos de muitas falhas, o seu uso pode ser mais custoso do que o com as instâncias *On-Demand*.

Como trabalhos futuros, pretende-se verificar a probabilidade de ocorrerem revogações reais em máquinas *Spot* utilizando *AWS ParallelCluster*, além de analisar a possibilidade de quando houver falha durante a execução, a aplicação ser iniciada em uma instância *On-Demand*, ao invés de ser iniciada novamente em uma instância *Spot*. Além do mais, é considerado analisar a escalabilidade do BRAMS nos sistemas abordados considerando o uso de uma quantidade maior de nós computacionais. Também pretende-se analisar diferentes configurações de *cluster* criados a partir do *AWS ParallelCluster*, como forma de reduzir os custos. Para isso, pretendemos, por exemplo, analisar a possibilidade de usar outras instâncias para o *head node*. Finalmente, é pretendido investigar mais a fundo as eventuais variações de desempenho obtidas, o que levou a *speedups* elevados. Um caminho para tal análise é verificar a contenção de acesso à memória e aos recursos de entrada e saída, quando a aplicação é executada em um único nó na nuvem.

Agradecimentos

Os autores agradecem o Laboratório Nacional de Computação Científica (LNCC/MCTI, Brasil) por fornecer recursos de HPC do supercomputador SDumont, que contribuíram para os resultados da pesquisa relatados neste artigo. URL: http://sdumont.lncc.br. Este trabalho também foi desenvolvido utilizando recursos do Projeto Universal/CNPq número 404087/2021-3.

Referências

Ahrens, J. H. and Dieter, U. (1974). Computer methods for sampling from gamma, beta, poisson and bionomial distributions. *Computing*, 12(3):223–246.

Amazon Web Service, I. (2023). *Amazon Elastic Compute Cloud: Manual do usuário para instâncias do Linux*.

Amazon Web Services, I. (2023a). *AWS Fault Injection Simulator: User Guide*.

Amazon Web Services, I. (2023b). *AWS ParallelCluster: AWS ParallelCluster User Guide (v3)*.

Benacchio, T., Bonaventura, L., Altenbernd, M., Cantwell, C. D., Düben, P. D., Gillard, M., Giraud, L., Göddeke, D., Raffin, E., Teranishi, K., et al. (2021). Resilience and fault tolerance in high-performance computing for numerical weather and climate prediction. *The International Journal of High Performance Computing Applications*, 35(4):285–311.

Bez, J. L., Carneiro, A. R., Pavan, P. J., Girelli, V. S., Boito, F. Z., Fagundes, B. A., Osthoff, C., da Silva Dias, P. L., Méhaut, J.-F., and Navaux, P. O. (2020). I/o performance of the santos dumont supercomputer. *The International Journal of High Performance Computing Applications*, 34(2):227–245.

Bourhnane, S. and Abid, M. R. (2020). High-performance computing as a cloud computing service. *International Journal of Advanced Trends in Computer Science and Engineering*.

Carreno, E. D., Roloff, E., and Navaux, P. O. (2015). Challenges and solutions in executing numerical weather prediction in a cloud infrastructure. *Procedia Computer Science*, 51:2832–2837.

Carreno, E. D., Roloff, E., and Navaux, P. O. (2016). Towards weather forecasting in the cloud. In *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, pages 659–663. IEEE.

Castro, P., Ishakian, V., Muthusamy, V., and Slominski, A. (2017). Serverless programming (function as a service). In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 2658–2659. IEEE.

de Araujo, L., Charão, A., Lima, J. V., and de Campos Velho, H. (2020). Análise de uma aplicaçao de modelagem atmosférica em nuvem e em contêineres utilizando rastros. In *Anais Estendidos do XXI Simpósio em Sistemas Computacionais de Alto Desempenho*, pages 54–61. SBC.

Fazenda, A. L., Rodrigues, E. R., Tomita, S. S., Panetta, J., and Mendes, C. L. (2012). Improving the scalability of an operational scientific application in a large multi-core cluster. In *Computer Systems (WSCAD-SSC), 2012 13th Symposium on*, pages 126–132. IEEE.

Freitas, S., Longo, K., Silva Dias, M., Chatfield, R., Silva Dias, P., Artaxo, P., Andreae, M., Grell, G., Rodrigues, L., Fazenda, A., et al. (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.

Freitas, S. R., Panetta, J., Longo, K. M., Rodrigues, L. F., Moreira, D. S., Rosario, N. E., Silva Dias, P. L., Silva Dias, M. A., Souza, E. P., Freitas, E. D., et al. (2017). The brazilian developments on the regional atmospheric modeling system (brams 5.2): an integrated environmental model tuned for tropical areas. *Geoscientific Model Development*, 10(1):189–222.

Freitas, S. R., Rodrigues, L. F., Panetta, J., Longo, K., Moreira, D., Freitas, E., Longo, M., Fazenda, A., Fonseca, R., Stockler, R., and Camponogara, G. (2016). *Description of the model input namelist parameters*. CPTEC/INPE, São Paulo, Brasil.

Guedes, T., Jesus, L. A., Ocaña, K. A., Drummond, L. M., and de Oliveira, D. (2020). Provenance-based fault tolerance technique recommendation for cloud-based scientific workflows: a practical approach. *Cluster Computing*, 23:123–148.

He, Q., Zhou, S., Kobler, B., Duffy, D., and McGlynn, T. (2010). Case study for running hpc applications in public clouds. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 395–401.

Koop, M. and Raman, K. (2021). Numerical weather prediction on aws graviton2.

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

Montes, D., Añel, J. A., Wallom, D. C., Uhe, P., Caderno, P. V., and Pena, T. F. (2020). Cloud computing for climate modelling: Evaluation, challenges and benefits. *Computers*, 9(2):52.

Netto, M. A., Calheiros, R. N., Rodrigues, E. R., Cunha, R. L., and Buyya, R. (2018). Hpc cloud for scientific and business applications: taxonomy, vision, and research challenges. *ACM Computing Surveys (CSUR)*, 51(1):1–29.

Pielke, R. A., Cotton, W., Walko, R. e. a., Tremback, C. J., Lyons, W. A., Grasso, L., Nicholls, M., Moran, M., Wesley, D., Lee, T., et al. (1992). A comprehensive meteorological modeling system—rams. *Meteorology and Atmospheric Physics*, 49(1-4):69–91.

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.

Sousa, W. P., Soares, F. M., Brum, R. C., Figueiredo, M., Melo, A. C., de Castro, M. C. S., and Bentes, C. (2023). Biological sequence comparison on cloud-based gpu environment. In *High Performance Computing in Clouds: Moving HPC Applications to a Scalable and Cost-Effective Environment*, pages 239–263. Springer.

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.

Xu, X., Mo, R., Dai, F., Lin, W., Wan, S., and Dou, W. (2019). Dynamic resource provisioning with fault tolerance for data-intensive meteorological workflows in cloud. *IEEE Transactions on Industrial Informatics*, 16(9):6172–6181.

Yoo, A. B., Jette, M. A., and Grondona, M. (2003). Slurm: Simple linux utility for resource management. In *Workshop on job scheduling strategies for parallel processing*, pages 44–60. Springer.