Avaliação da Biblioteca SCR em Instâncias AWS Spot Utilizando a Ferramenta HPC@Cloud

João Gabriel Feres[1], **Vanderlei Filho**[1], **Márcio Castro**[1]

[1]Universidade Federal de Santa Catarina (UFSC)
Florianópolis/SC

joao.feres@grad.ufsc.br, vanderlei.filho@proton.me,
marcio.castro@ufsc.br

Resumo. *A computação em nuvem tem sido útil para a Computação de Alto Desempenho (HPC), pois permite que experimentos em larga escala possam ser realizados sem a necessidade de aquisição de infraestruturas físicas de alto custo financeiro. Instâncias do tipo "spot" têm sido oferecidas pelos provedores de nuvem, as quais possuem baixo custo mas podem ser tomadas do usuário a qualquer momento. Esse artigo apresenta um estudo comparativo da biblioteca Scalable Checkpoint/Restart (SCR) com outras existentes para prover tolerância a faltas a aplicações de HPC em clusters de instâncias spot.*

1. Introdução

O mercado de computação em nuvem tem se aproximado da comunidade de Computação de Alto Desempenho (HPC) por meio da promoção de instâncias que possuem processadores de alto desempenho, *Graphics Processing Units* (GPUs) e redes de alto desempenho (e.g., *Infiniband*). Um dos produtos cuja integração com o HPC tem sido estudada são as instâncias *spot*, que historicamente surgiram como forma de utilizar recursos computacionais ociosos. Elas são instâncias mais baratas que as instâncias tradicionais (*on-demand*) mas podem ser tomadas do usuário a qualquer momento.

A ferramenta HPC@Cloud[1] [Munhoz and Castro 2023] foi desenvolvida para facilitar a criação de *clusters* e a execução de aplicações de HPC em nuvens públicas. Para tolerar faltas, a ferramenta também inclui um mecanismo que permite recuperar automaticamente instâncias *spot* caso sejam terminadas. Este artigo apresenta uma avaliação da biblioteca *Scalable Checkpoint/Restart* (SCR) [Moody et al. 2010] na ferramenta HPC@Cloud, comparando-a com outros dois mecanismos de tolerância a faltas existentes: *User Level-Fault Mitigation* (ULFM) [Bland et al. 2013] e *Berkeley Lab Checkpoint/Restart* (BLCR) [Hargrove and Duell 2006]. Para a realização dos experimentos, utilizou-se uma aplicação, doravante denominada HEAT, que resolve equações de difusão de calor de Laplace baseada em um Método de Diferenças Finitas (FDM) alternativo denominado *Forward-Time Central Space* (FTCS). Os resultados sugerem que o SCR pode apresentar melhor desempenho que os outros mecanismos semelhantes.

2. Tolerância a faltas e a Biblioteca SCR

Uma das técnicas mais conhecidas de tolerância a faltas é o *Checkpoint/Restart* (CR), que se resume a gravar estados da aplicação (*checkpoints*) para que, em caso de falta, a execução retorne dele, e não tenha que recomeçar. O CR pode ser periódico ou preemptivo, ou seja, os dados podem ser gravados periodicamente ou em situações específicas,

[1]https://github.com/lapesd/hpcac-toolkit/tree/hpc-toolkit-v2

Tabela 1. Instâncias EC2 da AWS utilizadas.

Tipo da Instância	vCPUs	Tipo de Armazenamento	Largura de Banda da Rede
t3.2xlarge	8	EBS	Até 5 Gbps
c5.2xlarge	8	EBS	Até 12.5 Gbps
c6i.2xlarge	8	EBS	Até 25 Gbps
i3.2xlarge	8	1x1900 NVMe SSD	Até 10 Gbps

como em avisos do sistema. A tolerância a faltas é importante no contexto de instâncias *spot*, pois a AWS pode revogar a instância durante a execução de uma aplicação de HPC.

A biblioteca SCR permite a aplicações MPI realizarem CR em *clusters* com armazenamento distribuído, e pode realizar diversas tarefas, como transferir dados assincronamente ao *Network File System* (NFS), se suportado, e rastreamento automatizado e reinicialização a partir do ponto de verificação mais recente. Além disso, disponibiliza comandos que são invocados de *batch job scripts*. Outro diferencial do SCR é a existência de "esquemas de redundância", que são formas tolerantes a faltas de armazenar dados não salvos no NFS entre os nós, de modo que seja reduzida a frequência de salvamento dos dados no NFS. O SCR define como "grupo de falta" um conjunto de processos que possui maior probabilidade de falta, como processos pertencentes a um mesmo nó. Há quatro esquemas de redundância, os quais definem onde e como os *checkpoints* são salvos. O esquema SINGLE é o mais básico, onde o *checkpoint* é escrito apenas no armazenamento local do nó ao qual o processo MPI pertence. Os demais esquemas oferecem alguma forma adicional de redundância. O PARTNER guarda uma cópia do *checkpoint* em outro grupo de falta. O esquema XOR guarda dados de paridade "XOR" num conjunto de processos de outros grupos de falta. Por fim, o esquema RS calcula uma codificação Reed-Solomon a partir de um conjunto de processos de diferentes grupos de faltas, e os dados de codificação são armazenados entre o conjunto.

3. Método de Análise

3.1. Ambiente Experimental

Os experimentos foram realizados em diferentes *clusters* homogêneos de oito nós criados com a ferramenta HPC@Cloud utilizando as instâncias da AWS mostradas na Tabela 1. Para acelerar a criação dos *clusters* foi preparada uma *Amazon Machine Image* (AMI)[2] com GCC 11.4.1, Open MPI 5.1.0 e SCR v3.0.1. Foram usadas instâncias *on-demand*, cujas diferenças em relação às *spot* sendo seu valor maior e a impossibilidade da retomada das instâncias pelo provedor, a fim de melhorar a reprodutibilidade dos resultados.

3.2. Aplicação

A aplicação alvo utilizada nos experimentos, introduzida na Seção 1, foi desenvolvida em um trabalho anterior [Munhoz and Castro 2023]. No presente trabalho, o código-fonte[3] da aplicação foi adaptado para funcionar com SCR.

Para os experimentos, foram fixados o tamanho da matriz e a quantidade de iterações. Foi usada uma região quadrada de tamanho 2048x2048 com 200 iterações e um processo MPI para cada vCPU da instância utilizada. Foram simuladas duas faltas no *cluster* durante a execução dos experimentos: a primeira em 1/3 e a segunda em 2/3

[2]https://docs.aws.amazon.com/pt_br/AWSEC2/latest/UserGuide/AMIs.html
[3]https://github.com/Prog-in/jacobi-method

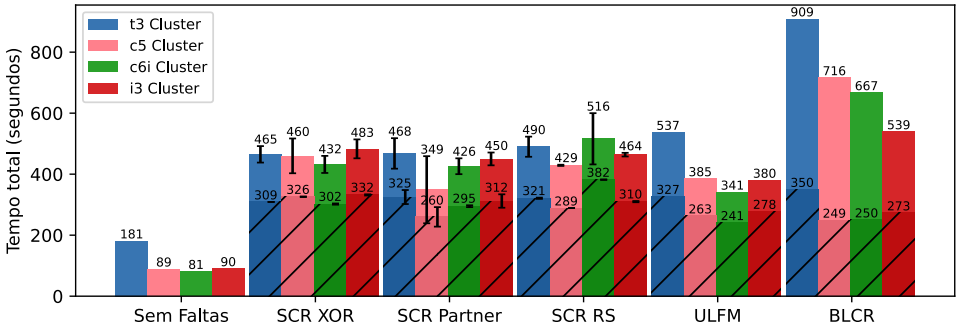

Figura 1. Tempos de execução da aplicação HEAT com diferentes mecanismos de CR e duas faltas.

do total de iterações, ambas feitas por meio de chamadas automatizadas da aplicação à *aws cli*. As iterações onde ocorrem as faltas são as mesmas usadas no trabalho anterior [Munhoz and Castro 2023], de onde foram extraídos os dados referentes às outras bibliotecas de CR (ULFM e BLCR). Ainda, os *checkpoints* foram realizados a cada 20 iterações. Para simplificar e deixar as testagens mais objetivas, definiu-se que apenas uma falta ocorre por vez e que faltas não ocorrem durante a escrita e leitura de *checkpoints*.

3.3. Configurações do SCR

Foram avaliados os esquemas de redundância descritos na Seção 2, a fim de analisar o desempenho de cada um. O esquema SINGLE não foi usado pois não asseguraria os dados não salvos no NFS caso um nó caísse. O parâmetro `SCR_FLUSH`, que indica o número de *checkpoints* feitos entre cópias para o NFS, foi fixado em 3, `SCR_FLUSH_ASYNC` em 1 para habilitar a escrita assíncrona no NFS, `SCR_FLUSH_TYPE` em "PTHREAD", `SCR_COPY_TYPE` em "FILE", `SCR_PREFIX_SIZE`, que representa a quantidade de *datasets* armazenados localmente em cada nó, em 1, e `SCR_CACHE_SIZE`, que indica a quantidade de *datasets* armazenados no NFS, em 1. Por fim, `SCR_SET_SIZE`, que indica o tamanho do conjunto de redundância, usado pelas estratégias XOR e RS, foi fixado em 8. Outras configurações do SCR não explicitadas aqui possuem valor padrão.

4. Resultados Experimentais

A Figura 1 apresenta os resultados obtidos com a aplicação HEAT. As áreas hachuradas indicam tempo ocioso, ou seja, o tempo gasto realizando *checkpoints*, restaurando instâncias e esperando as instâncias restauradas serem recriadas. Os resultados foram obtidos calculando-se a média aritmética de cinco execuções.

Os resultados mostram que os esquemas de redundância apresentam desempenhos relativamente diferentes. Ainda, quando comparados com as outras estratégias, apresentaram resultados mais homogêneos em relação aos diferentes *clusters*, o que aumenta a previsibilidade dos resultados. Também, percebe-se que todos os esquemas de redundância do SCR apresentaram melhores resultados que a biblioteca BLCR, e tendem a ter tempos de execução parecidos ou até mesmo menores que a biblioteca ULFM.

A Figura 2 compara os tempos médios de execução das funções SCR em relação ao seu tempo total quando executando o HEAT com o SCR PARTNER em um *cluster* com oito nós t3.2xlarge. O objetivo desse experimento é analisar o custo computacional das funções internas do SCR. Os resultados foram derivados da média aritmética de 5

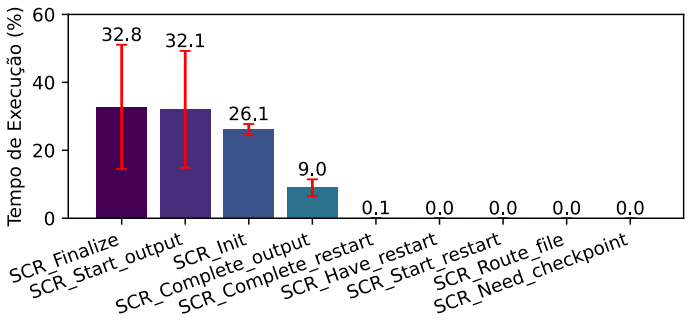

Figura 2. Comparação entre os tempos médios das funções do SCR em comparação ao tempo total das funções do SCR.

execuções. Neste experimento, ao invés de utilizar a função `SCR_Need_checkpoint`, foi feita a verificação das iterações manualmente devido ao seu alto tempo de execução. Uma investigação mais detalhada desta função será necessária para entender o motivo deste alto sobrecusto. Foi constatado que as quatro funções de maior custo do SCR são `SCR_Finalize`, `SCR_Start_output`, `SCR_Init` e `SCR_Complete_output`. Como `SCR_Init` é chamada apenas quando a aplicação é iniciada (ou reiniciada) e `SCR_Finalize` quando a aplicação é terminada, seus tempos médios serão proporcionalmente menores conforme a quantidade de iterações aumenta. O tempo total das funções SCR em comparação com o tempo total da aplicação HEAT foi de 9.84%.

5. Conclusão

Nesse artigo, foi apresentada uma avaliação da biblioteca SCR na ferramenta HPC@Cloud. Os experimentos foram realizados na AWS e foram avaliados os diferentes esquemas de redundância que o SCR suporta. Ainda, foram comparados os tempos de execução dos esquemas de redundância SCR com os de outras bibliotecas. Os resultados sugerem que o SCR pode ser tão, se não mais performático que as outras bibliotecas e estratégias analisadas. Como trabalhos futuros, pretende-se realizar experimentos de maior escala e testar a biblioteca SCR com outras aplicações de HPC, além de avaliar outras métricas, como uso de CPU, memória e taxa de I/O em disco, a fim de compreender mais profundamente como a biblioteca SCR se comporta.

Referências

Bland, W., Bouteiller, A., Herault, T., Bosilca, G., and Dongarra, J. (2013). Post-failure recovery of mpi communication capability: Design and rationale. *The International Journal of High Performance Computing Applications*, 27(3):244–254.

Hargrove, P. H. and Duell, J. C. (2006). Berkeley lab checkpoint/restart (blcr) for linux clusters. In *Journal of Physics: Conference Series*, volume 46, page 494. IOP Publishing.

Moody, A., Bronevetsky, G., Mohror, K., and De Supinski, B. R. (2010). Design, modeling, and evaluation of a scalable multi-level checkpointing system. In *SC'10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11. IEEE.

Munhoz, V. and Castro, M. (2023). Enabling the Execution of HPC Applications on Public Clouds with HPC@Cloud Toolkit. *Concurrency and Computation: Practice and Experience*, pages 1–19.