Investigando Fatores de Desempenho em Dispositivos Persistentes

Lucas Spagnol[1], **Bruno Honorio**[1], **Edgar Galvão**[1], **Emilio Francesquini**[2], **Alexandro Baldassin**[1]

[1]Universidade Estadual Paulista (UNESP) – Rio Claro, SP

{lucas.b.spagnol, bruno.honorio, edgar.galvao, alexandro.baldassin}@unesp.br

[2]Universidade Federal do ABC (UFABC) – Santo André, SP

e.francesquini@ufabc.edu.br

Abstract. *Persistent Memory is an emerging technology that combines the characteristics of DRAM and SSD, enabling its application in a variety of tasks. However, some issues may arise regarding its functionality and performance when tested on a machine equipped with it, especially with respect to NUMA devices and cache operations. In light of this, this paper presents some insights that seek to uncover how these factors impact the performance of persistent devices.*

Resumo. *A Memória Persistente é uma tecnologia emergente que combina as características da DRAM e do SSD, possibilitando sua aplicação em uma variedade de tarefas. No entanto, alguns reveses relacionados ao seu funcionamento e desempenho quando testada em uma máquina equipada com ela podem surgir, especialmente no que diz respeito às máquinas NUMA e às operações em cache. Diante disso, apresentamos algumas observações que buscam revelar como esses fatores impactam o desempenho de dispositivos persistentes.*

1. Introdução

A Memória Persistente (PM) representa uma inovação tecnológica que almeja combinar as propriedades da memória volátil, tais como a baixa latência e o endereçamento a byte, com os benefícios da persistência da memória não-volátil. Em 2018, a Intel lançou comercialmente a Intel Optane, fundamentada na tecnologia 3D Xpoint [Seltzer et al. 2018]. A partir disso, uma série de estudos foram conduzidos com o objetivo de analisar tanto o desempenho quanto a funcionalidade desta tecnologia.

Ao executar os *benchmarks* descritos no artigo [Islam and Dai 2023], um *framework* estado-da-arte para análise de grafos dinâmicos, observou-se que o SSD (*Solid State Drive*) exibia um desempenho superior ao da PM. Os resultados obtidos por nós mostraram o SSD alcançando um tempo de execução equivalente à metade do tempo registrado para a PM. Notou-se, posteriormente, que o principal fator por trás deste comportamento é o sistema de cache empregado pelo sistema operacional. Outro ponto importante nos experimentos refere-se ao uso de uma arquitetura NUMA (Non-Uniform Memory Access) com dois nós. Nossos resultados experimentais mostram que há uma diferença de desempenho de 15% a depender se o acesso é intra- ou inter-nós.

Este artigo procura elucidar alguns fatores chaves que devem ser considerados na análise de desempenho com memórias persistentes. O primeiro deles refere-se ao fator NUMA, já conhecido no âmbito de memórias voláteis, mas ainda não explorado em

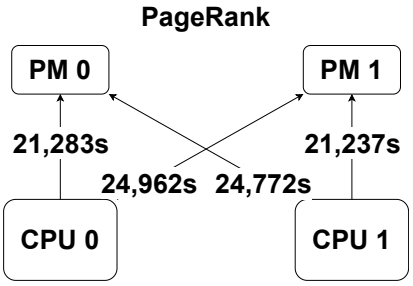


Figura 1. Tempo necessário para a execução do *Page Rank*, em segundos.

memórias persistentes. O segundo fator diz respeito ao esquema de cache utilizado pelo sistema operacional, o que pode mascarar o desempenho observado nos experimentos. Assim, pretende-se com esse breve estudo chamar a atenção dos pesquisadores que fazem experimentos com dispositivos persistentes sobre a importância de configurar corretamente o ambiente experimental.

2. Efeito NUMA em Memórias Persistentes

Antes de abordar o problema em questão, é essencial compreender o funcionamento da memória persistente. A máquina empregada para os testes é equipada com dois processadores da *Intel* e oito módulos de memória persistente *Optane DC*, também da *Intel*. A memória persistente é conectada ao barramento do processador, o mesmo utilizado para a DRAM. Dessa forma, os módulos de memória persistente estão distribuídos entre os barramentos dos processadores, com quatro módulos alocados para cada processador. Na *BIOS*, esses módulos são configurados para formar uma única unidade de memória, resultando em dois nós de memória persistente para o sistema operacional, os quais chamamos de $PM0$ e $PM1$.

Durante experimentos com o DGAP [Islam and Dai 2023], um framework para processamento de grafos dinâmicos que utiliza memória persistente, se notou um comportamento anômalo de acesso a essa memória e se levantou a suspeita que pudesse ser devido ao fator NUMA. Para verificar essa suposição, foi empregada a ferramenta *Taskset*[1], que permite selecionar quais *threads* do processador estarão disponíveis para o *benchmark*. Dessa forma, o teste foi realizado na CPU0, acessando ambas as memórias, e posteriormente na CPU1, também acessando ambas as memórias $PM0$ e $PM1$. Com essa abordagem, foi possível observar que, ao acessar a memória conectada ao barramento do outro processador, o desempenho diminuiu aproximadamente 15%, conforme ilustrado na Figura 1. Os testes foram feitos utilizando o *benchmark* de análise de grafos *Page Rank*. Essas observações revelam a importância do desenvolvimento de métodos eficientes para alocação de memória em máquinas NUMA com PM.

3. Desempenho do SSD vs. PM

Para abordar o problema do SSD estar mais rápido que a PM, a primeira ação executada consistiu na desativação do *DAX* na memória persistente. O *DAX* possibilita o acesso direto à memória persistente (PM).A desativação do DAX fez com o que sistema operacional utilizasse o mesmo esquema de bufferização (page caching) utilizado no SSD, aumentando o desempenho da PM. É importante observar que ao utilizar o esquema de

[1]https://man7.org/linux/man-pages/man1/taskset.1.html

bufferização, a propriedade de *durabilidade imediata* [Baldassin et al. 2021] da PM é perdida, ou seja, os últimos dados escritos não necessariamente terão sido persistidos.

No caso do SSD, o acesso geralmente é feito por uma API (*Application Programming Interface*) de arquivos. Essa API comumente realiza algum tipo de bufferização para acelerar o acesso aos dados, uma vez que este acesso é notavelmente mais lento do que o acesso a DRAM. Ou seja, para garantir a durabilidade imediata dos dados é necessário forçar que os dados armazenados no buffer sejam escritos no dispositivo.

Assim, para uma comparação justa entre os dispositivos, é necessário evitar que o SSD utilize a bufferização. Uma primeira alternativa empregada foi montar o sistema de arquivos da SSD com a opção *flag -sync*. Isso realmente fez com que as escritas fossem persistidas diretamente no disco, mas não alterou o comportamento das leituras (o tempo de execução com o DGAP permaneceu o mesmo, visto que ele basicamente só fazia leituras). Outra opção que adotada foi alterar a configuração do sistema operacional (Linux) para eliminar a bufferização de escritas, modificando as variáveis *vm.dirty_background_ratio* e *vm.dirty_ratio* do arquivo */etc/sysctl.conf*[2]. Essas variáveis definem a quantidade de memória DRAM que será destinada para a utilização da *page cache*.

Houve bastante dificuldade para evitar a bufferização das leituras realizadas pelo sistema operacional, uma vez que não há um procedimento bem documentado para isso. Uma alternativa encontrada foi a utilização do seguinte comando:

```
sudo sync; echo 1 > /proc/sys/vm/drop_caches
```

forçando o sistema operacional a limpar toda a bufferização realizada. Portanto este comando é necessário caso seja necessário também levar em consideração o tempo de acesso de leituras ao SSD. Note, no entanto, que essa alternativa não é uma solução completa, pois ela exige executar o comando para limpar a bufferização regularmente. Para fins práticos, é suficiente desabilitar a bufferização apenas de escrita.

4. Experimentos utilizando a API de arquivos

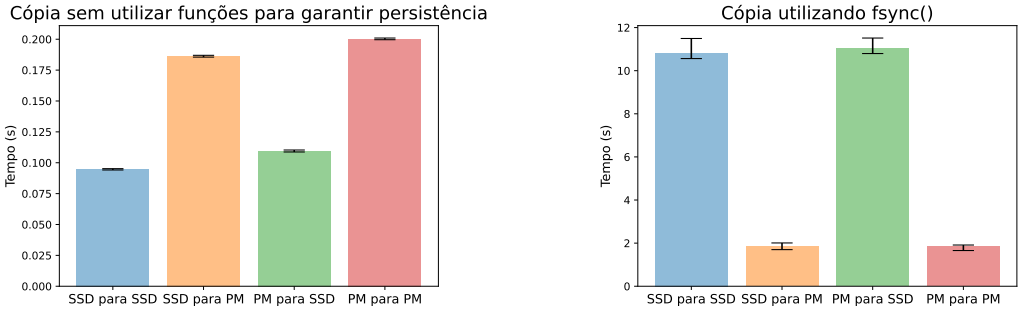
Para investigar a discussão anterior em um cenário específico, foi desenvolvido um *benchmark* que copia um arquivo de um diretório para outro em blocos. Este *benchmark* foi implementado utilizando as funções de sistema de arquivos da linguagem C, especificamente, `fopen(), fread(), fwrite()`. Durante a cópia do arquivo, o programa registra o tempo decorrido e exibe este valor ao final do processo.

Ao empregar exclusivamente as funções mencionadas, observou-se (veja Figura 2a) que o SSD apresentou um desempenho superior ao da PM. Esta superioridade do SSD pode ser atribuída ao fato de o sistema operacional realizar a bufferização de escrita na DRAM e, posteriormente, salvar os arquivos no SSD. Em contraste, na PM, devido à ativação do DAX, as operações de escrita são executadas diretamente na PM.

Para investigar métodos de evitar o uso da bufferização sem a necessidade de alterar configurações na máquina, foram testadas três chamadas da API: `fflush()` e `fsync()`[3]. O objetivo era determinar se essas funções influenciariam os resultados obtidos. A função `fflush()` limpa o *buffer* da aplicação (no nível da (glibc)), enviando para o sistema operacional os dados que serão escritos na memória. Por outro lado,

[2]https://man7.org/linux/man-pages/man5/sysctl.conf.5.html

[3]É possível encontrar os *benchmarks* no repositório:https://github.com/LucasBastelli/block-copy.git



(a) Cópia comum. **(b) fsync().**

Figura 2. Tempo(s) de cópia de um arquivo de 100MB entre os dispositivos.

`fsync()` faz uma chamada ao sistema operacional para forçar a escrita dos dados no dispositivo.

São apresentados os resultados experimentais da comparação dos tempos necessários para a cópia de arquivos de 100MB entre os dispositivos de armazenamento utilizando a função `fsync()`. Cada experimento foi repetido 25 vezes para o cálculo da média. O cálculo do intervalo de confiança de 95% foi efetuado com a técnica de Bootstrap com 10.000 reamostragens.

Em primeiro lugar, nossos experimentos mostraram que a chamada `fflush()` não alterou os resultados. Ou seja, a bufferização em nível de aplicação não parece estar influenciando no ganho de desempenho. No entanto, a chamada `fsync()` resultou em um aumento no tempo de execução, tornando o SSD significativamente mais lento quando comparado à memória persistente (PM), como é possível observar na Figura 2b. Também foram realizados testes similares usando a API de mapeamento de memória (`mmap`) e a chamada `msync()`, com resultados similares ao apresentado na Figura 2b. Estes resultados mostram que é necessário garantir a escrita dos dados na SSD quando o desempenho é comparado com a PM.

5. Conclusão

Observou-se que o efeito da bufferização no SSD resulta em um desempenho aparentemente superior ao real, o que pode levar à percepção inicial de que a Memória Persistente é “lenta”. Portanto, é essencial adotar métodos que evitem o uso da bufferização, uma vez que ela impede a persistência dos dados escritos. Além disso, notou-se que o efeito NUMA influencia o desempenho da Memória Persistente, tornando-se necessário minimizar seu uso quando o objetivo é otimizar o desempenho.

Agradecimentos. Os autores agradecem à Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), processos nº 2023/04969-8, 2023/04971-2, 2018/15519-5 e 2023/10128-6 pelo apoio a este trabalho.

Referências

Baldassin, A., Barreto, J. a., Castro, D., and Romano, P. (2021). Persistent memory: A survey of programming support and implementations. *ACM Comput. Surv.*, 54(7).

Islam, A. A. R. and Dai, D. (2023). Dgap: Efficient dynamic graph analysis on persistent memory. In *Proc. of the SC'23*.

Seltzer, M., Marathe, V., and Byan, S. (2018). An NVM Carol: Visions of NVM Past, Present, and Future. In *Proc. of the ICDE'18*, pages 15–23.