Primeiras Experiências com a Programação de Estruturas de Dados Persistentes

Lucas Bastelli, Alexandro Baldassin

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

{lucas.b.spagnol, alexandro.baldassin}@unesp.br

Abstract. *Persistent memory is the latest storage technology. Unlike secundary storages devices (HD and SSD), its high access speed, lower latency and low granularity, allows direct connection with the processor's bus. But, to use this new technology, it is necessary new programming models to secure that data is always updated consistently. In this context, this paper shows the use of Intel's Persistent Memory Development Kit (PMDK) to build a singly linked list data structure. We discuss the challenges of programming for persistent rather than volatile memory and present the first results comparing the performance of commom DRAM and Intel Optane DC.*

Resumo. *Memória persistente é uma das mais novas tecnologias em armazenamento. Contrário aos dispositivos para armazenamento secundário (HD e SSD), sua maior velocidade de acesso, menor latência e granularidade baixa, possibilitam a conexão direta com o barramento do processador. Porém, para a utilização dessa nova tecnologia, também são necessários novos meios de programação para garantir a consistência dos dados. Neste contexto, este trabalho mostra a utilização da biblioteca Persistent Memory Development Kit (PMDK) da Intel para o desenvolvimento de uma lista ligada simples. São discutidos os desafios com a programação para memória persistente e apresentados resultados iniciais que comparam o desempenho de memória volátil (DRAM) com a Intel Optane DC.*

1. Introdução

Tradicionalmente, dispositivos de armazenamento em sistemas computacionais são divididos entre primário e secundário. O armazenamento primário (tipicamente DRAM) permite acesso rápido aos dados, porém sua capacidade geralmente é restrita e os dados são voláteis, ou seja, são perdidos quando o fornecimento de energia é interrompido. No segundo caso (tipicamente HD ou SSD) os dados podem ser mantidos por tempo indeterminado, mas a velocidade de acesso geralmente é ordens de grandezas mais lento e portanto não são conectados diretamente no barramento do processador [Hennessy and Patterson 2017]. Os recentes avanços na indústria de semicondutores têm permitido o desenvolvimento de novas tecnologias para armazenamento persistente com características de armazenamento primário: os dispositivos podem ser conectados diretamente ao barramento do processador, permitindo operações de leituras e escritas na granularidade de byte, com a vantagem da persistência. Tais dispositivos tem sido chamados de *Memória Persistente* (PM – *Persistent Memory*) [Baldassin et al. 2021]. Comercialmente, a Intel lançou recentemente o Optane DC, baseado na tecnologia 3D XPoint [Seltzer et al. 2018].

A introdução da PM faz com que a forma que o código é escrito seja diferente da forma que tradicionalmente é feita para dispositivos secundários. Como o acesso é feito diretamente na PM, caso haja uma falha durante o acesso (acabe a energia, por exemplo), uma estrutura de dados pode estar em um estado inconsistente no momento da reinicialização do sistema. Este trabalho mostra como utilizar a biblioteca PMDK da Intel [Scargall 2020] para desenvolver uma estrutura de dados do tipo lista ligada simples. Em particular, a PMDK oferece a abstração de *transações*, facilitando o gerenciamento de erros causados for falhas que forcem a reinicialização do sistema. Além da apresentar os desafios de se programar com PM, também são discutidos resultados iniciais de desempenho comparando PM, DRAM e SSD.

2. Intel PMDK

O PMDK é composto por diversos módulos, com diferentes níveis de abstração. O mais comum deles, e usado neste trabalho, é conhecido como *libpmemobj* para a linguagem C. Este módulo fornece suporte para: i) trabalhar com *pool* de memória persistente; ii) alocação de memória persistente; e iii) atualização de dados persistentes de forma consistente por meio de transações. Antes de realizar qualquer operação na PM, um *pool* de memória precisa ser criado/aberto. Uma vez aberto, a PM é acessada através de um *objeto raiz*, a partir do qual todos os outros objetos na PM são acessados. A alocação de memória persistente geralmente é feita através de transações. Isso ocorre porque entre a chamada para alocação de memória persistente e o armazenamento do endereço alocado, podem haver falhas (como queda de energia, por exemplo). Neste caso, poderia haver um vazamento de memória persistente.

Finalmente, o suporte para transações é o principal aspecto fornecido pelo PMDK. Uma transação garante que todas as alterações realizadas dentro dela, ou são completadas e ditas persistentes (efetivadas ou *committed*), ou então o sistema reverte as alterações e é como se nada tivesse acontecido (*rollback*). As transações, desta forma, possibilitam a atualização de dados persistentes de forma consistente.

3. Implementando uma lista ligada persistente

Para implementar a lista ligada persistente [1], usou-se como base uma equivalente já desenvolvida para memória volátil. A partir disso, foi necessário adaptar o código para utilizar o PMDK. O código da Listagem 1 é usado para elucidar alguns conceitos descritos a seguir. A primeira grande diferença começa com a declaração das estruturas, apresentada entre as linhas 1 e 8. Ao contrário da versão volátil, é necessário que os ponteiros sejam persistentes. Uma das formas de especificar isso é utilizando a macro `TOID()`. Desta forma, `next` (linha 2) é um ponteiro persistente, assim como o ponteiro para o objeto raíz que aponta para a cabeça da lista, `head` (linha 7). No PMDK, um ponteiro persistente tem 128 bits e é composto por um identificador da *pool*, de 64 bits, e um ponteiro convencional, também de 64 bits.

Outra grande diferença está no conceito de *pool de memória persistente*, requerendo chamadas explícitas para seu gerenciamento (linhas 10 e 11). O conceito mais similar no contexto de memória volátil seria o do *heap* de memória. O *pool* é exposto como um arquivo presente no sistema de arquivos do sistema operacional e precisa ser

[1] O código pode ser acessado pelo seguinte link: https://github.com/LucasBastelli/IC_PMDK

explicitamente aberto/criado (linha 10). A seguir, é necessário acessar o *objeto raiz* (linha 11), através do qual todos os outros elementos do *pool* serão acessados.

```
struct entry{
  TOID(struct entry) next;
  val_t val;
 };

struct root{
  TOID(struct entry) head;
 };

PMEMobjpool *pop = pmemobj_open("mylist",...);
TOID(struct root) root = POBJ_ROOT(pop, struct root);

// inserir elemento no começo da lista
TX_BEGIN(pop){
  TOID(struct entry) entry;
  entry = TX_ALLOC(struct entry,sizeof(struct entry));
  D_RW(entry)->val = value;
  D_RW(entry)->next = D_RO(root)->head;
  TX_ADD(root);
  D_RW(root)->head = entry;
}TX_END
```

Listagem 1. Trechos de código com a lista ligada persistente.

As transações são usadas para alterar de forma consistente os dados persistentes. Um exemplo que insere um elemento no começo da lista é mostrado entre as linhas 14 e 21. É necessário especificar o *pool* no qual as operações da transação serão efetuadas, no caso o objeto `pop` (linha 14) aberto na linha 10. As linhas de 15 a 18 são responsáveis por alocar o novo nodo na memória persistente e inicializá-lo. Para dereferenciar ponteiros persistentes, geralmente a macro `D_RW` (linhas 17 e 18) é utilizada. Se o acesso é somente de leitura, a macro `D_RO` deve ser usada (linha 18). Finalmente, o ponteiro para a cabeça da lista é atualizado (linha 20). As transações no PMDK utilizam *undo logs* para garantir a consistência. O *undo log* armazena uma cópia prévia do dado alterado, de forma que esse valor possa ser restaurado em caso de alguma falha durante o processamento da transação (por exemplo, queda de energia). O programador é responsável por informar os dados que devem ser salvos no *undo log* através do `TX_ADD` (linha 19). Note que, caso isso não seja feito, pode ser que a cabeça da lista seja alterada (linha 20) mas, antes da transação terminar (linha 21), ocorra uma queda de energia. Nesse caso, quando o sistema for restaurado, a cabeça da lista vai apontar para um endereço inválido, já que a transação não terminou e portanto a alocação de memória foi revertida (linha 16), mas não a atribuição (linha 20).

4. Resultados Experimentais

São apresentados resultados iniciais de desempenho da memória persistente (Intel Optane DC) quando comparada com DRAM e SSD. O benchmark utilizado realiza, por 10 segundos, operações de inserção, remoção e busca numa estrutura de lista ligada simples. A máquina utilizada é um Intel Xeon gold 5317 com 128GB de DRAM. Os experimentos foram executados 10 vezes e os resultados da Figura 1 apresentam a vazão em operações por segundo (eixo Y) para diferentes tamanhos da estrutura de dados (eixo X). A taxa de atualização (operações de inserção e remoção) da lista ligada é de 50%.

Na Figura 1 é possível perceber que a PM ficou, em média, 4x mais lenta que a DRAM e 188x mais rápida que a SSD. Desta forma, a memória persistente consegue unir

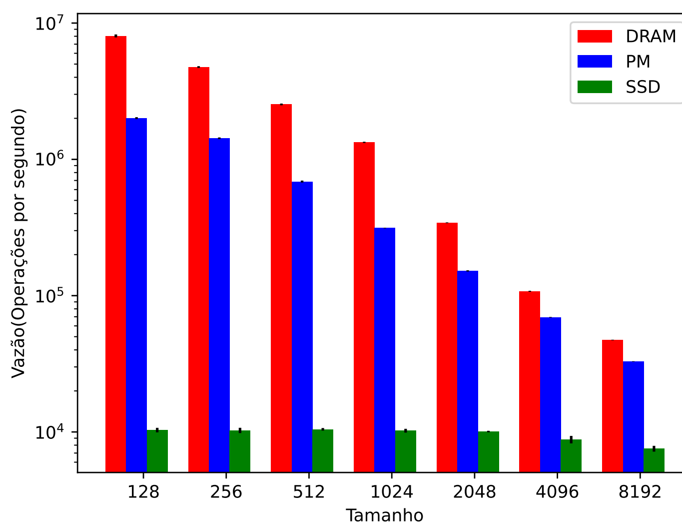

Figura 1. Comparação entre as velocidades.

a velocidade da memória RAM com a possibilidade de persistência dos dados como uma SSD. Também é possível observar a grande diferença de velocidade entre DRAM e PM em comparação ao SSD. Foi necessário utilizar um gráfico em escala logarítmica, pois as velocidades delas são muito maiores em comparação ao SSD.

5. Conclusão

Memória persistente (PM) apresenta uma série de vantagens em relação aos dispositivos de armazenamento secundário, como o HD e SSD. Entre elas a principal é a alta velocidade de acesso, permitindo a conexão direta ao barramento do processador. Neste artigo, analisou-se o uso da biblioteca PMDK para a programação de estruturas de dados persistentes. Os resultados iniciais indicam que a PM consegue um valor muito próximo ao da memória DRAM. Notou-se que a PM alia as principais características da memória DRAM e do SSD: a velocidade e a persistência dos dados.

Agradecimentos

Este trabalho teve apoio dos processos nº 2021/05440-5 e 2018/15519-5, Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP).

Referências

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

Hennessy, J. L. and Patterson, D. A. (2017). *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 6th edition.

Scargall, S. (2020). *Programming Persistent Memory - A Comprehensive Guide for Developers*. Apress, 1st edition.

Seltzer, M., Marathe, V., and Byan, S. (2018). An NVM Carol: Visions of NVM Past, Present, and Future. In *Proceedings of the IEEE 34th International Conference on Data Engineering (ICDE)*, pages 15–23.