Uma Revisão Sistemática sobre Estruturas de Dados em Dispositivos Persistentes Contemporâneos

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

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

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

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

e.francesquini@ufabc.edu.br

Resumo. *Memória persistente (PM) é uma tecnologia emergente que combina o acesso rápido aos dados com endereçamento por byte, além de oferecer grande capacidade de armazenamento persistente. Devido a essas características, a PM tem ganhado cada vez mais atenção, com diversos estudos sendo realizados para explorar seu uso e desempenho. Paralelamente, trabalhos sobre estruturas de dados (ED) têm evoluído para maximizar o aproveitamento das PMs. Isso inclui o desenvolvimento de EDs adaptadas para a persistência e transações atômicas. Este artigo revisa os trabalhos mais relevantes na área de memória persistente com foco em estruturas de dados, destacando aqueles que desenvolvem ou adaptam tais estruturas para uso na PM. Nossos estudos identificaram as principais características necessárias para o desenvolvimento de uma estrutura de dados voltada para PM, visando aprimorar o desempenho ao reduzir o número de operações de escrita por meio de otimizações de código e do uso de memória DRAM como cache. Além disso, enfatizamos a importância da garantia de persistência, uma vez que a memória persistente mantém os dados mesmo após o desligamento do sistema.*

1. Introdução

Computadores convencionalmente são divididos em armazenamento primário (memória DRAM) e secundário (HDs e SSDs). O armazenamento primário permite ao computador acessar rapidamente os dados, porém com pouco espaço de armazenamento e de forma volátil, ou seja, quando o computador é desligado, seja por falta de energia, erros, ou por comando do usuário, todos os dados armazenados nele são perdidos. Já no armazenamento secundário, é possível a retenção de dados mesmo após o desligamento do sistema. Em contrapartida, ao contrário da DRAM, esses dispositivos apresentam largura de banda pequena e tempos de resposta elevados quando comparados a ela, tornando-os inadequados para operações que exigem rapidez, o que os torna mais apropriados para armazenamento de dados [8].

Os recentes avanços na indústria de semicondutores viabilizaram o desenvolvimento de tecnologias inovadoras para armazenamento persistente [1]. Essas tecnologias unem as características da DRAM e dos HDs e SSDs, como altas taxas de transferência de dados e granularidade de byte, juntamente com a grande capacidade de armazenamento e persistência dos dados. Isso possibilita a realização de operações na Memória Persistente (PM) de forma rápida e persistente.

As estruturas de dados (ED) desempenham um papel fundamental nos sistemas computacionais. Elas permitem armazenar e processar informações de maneira eficiente

e organizada, sendo utilizadas desde caches para acelerar o acesso a dados e arquivos até para armazenar informações em bancos de dados. Com o avanço da memória persistente, torna-se necessário desenvolver ou adaptar as estruturas de dados para que elas possam aproveitar a melhoria de desempenho além, é claro, da persistência.

Contudo, ao integrar estruturas de dados com persistência, é imprescindível adotar medidas preventivas para assegurar a consistência dos dados, evitando assim a perda de dados e vazamentos de memória. Comumente, empregam-se transações para garantir a correta gravação dos dados [7]. Este trabalho tem como objetivo apresentar um levantamento atual da pesquisa sobre o desenvolvimento das estruturas de dados voltadas para a memória persistente. Busca-se responder questões sobre essas estruturas de dados em relação ao seu uso e funcionalidade, por meio de uma revisão sistemática de artigos publicados na área nos últimos 5 anos.

A análise dos trabalhos encontrados possibilita a elaboração de uma lista de requisitos essenciais que uma estrutura deve possuir para operar de maneira otimizada na memória persistente. Essas informações se mostram valiosas ao se considerar o desenvolvimento de uma nova estrutura de dados voltadas à PM. O objetivo principal deste artigo é identificar e empregar os requisitos essenciais na elaboração de estruturas de dados para memória persistente. Como contribuição, este artigo apresenta as estruturas de dados mais utilizadas na área de PM, bem como as otimizações aplicadas para melhorar o desempenho neste tipo de dispositivo. Além disso, são determinados os principais fatores que ainda não foram explorados na PM, destacando áreas potenciais para futuras pesquisas e desenvolvimento.

2. Contexto

As estruturas de dados exercem um papel crucial na computação, uma vez que possibilitam a representação organizada das informações na memória dos computadores. Algumas dessas estruturas, como é o caso da *hash table*, proporcionam ainda uma maior eficiência na busca de dados. Com o progresso das memórias persistentes que possuem endereçamento a *byte*, torna-se imprescindível a adaptação das estruturas de dados para essas memórias, de modo que possam ser utilizadas de maneira eficaz e com a capacidade de persistência. Diferentemente das estruturas empregadas na memória DRAM, as estruturas adaptadas para as memórias persistentes devem ser modificadas para serem resistentes a falhas.

Considere, por exemplo, uma lista encadeada que contém os valores 1, 3, 7 e 9, conforme ilustrado na Figura 1. Suponha que desejamos inserir o valor 5 nessa lista. Durante esse processo de inserção, se alguma falha ocorresse (por exemplo, queda de energia), poderíamos enfrentar duas situações problemáticas. A primeira situação (A) é a possibilidade do valor ser alocado na memória mas, por alguma falha (como falta de energia), não ser efetivamente inserido na lista. Isso resultaria em um vazamento de memória, onde um espaço ficará ocupado mas sem uso real. A segunda situação (B) ocorre quando o nó contendo o valor 3 é atualizado para apontar para o novo nó 5, mas o ponteiro do novo nó 5 não é ajustado para apontar para o nó com o valor 7 (novamente, alguma falha pode acontecer antes desta última escrita). Isso resultaria na perda de todos os dados subsequentes na lista, uma vez que o encadeamento dos nós é interrompido. Essas situações destacam a importância de garantir a integridade dos dados durante as operações.

A presente pesquisa não identificou revisões sistemáticas anteriores que abordas-

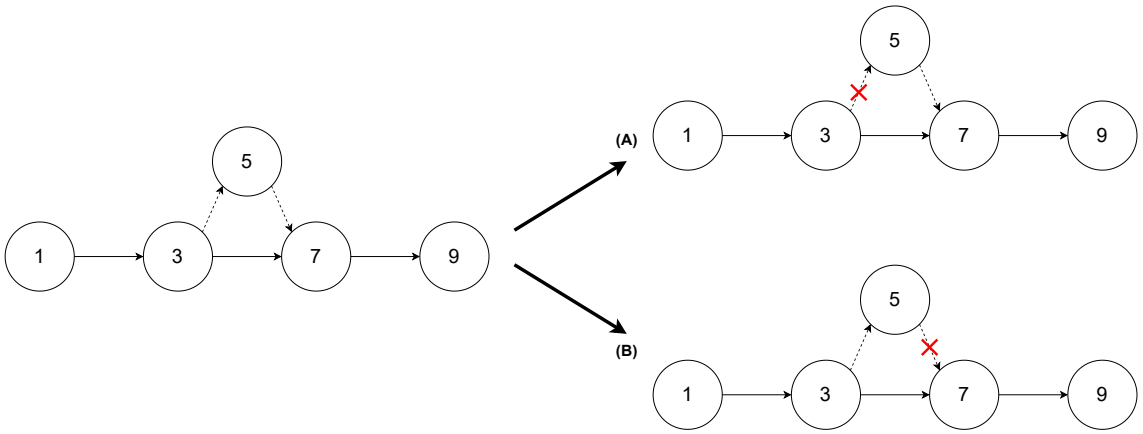

Figura 1. Casos de falhas na inserção de um elemento em uma lista encadeada.

sem questões como: quais são as estruturas de dados mais utilizadas, quantos trabalhos disponibilizam código aberto e quais otimizações foram implementadas para melhorar o desempenho na memória persistente. Essas questões são detalhadas na Seção 4. Este artigo analisa os aspectos mais recorrentes em trabalhos sobre estruturas de dados na memória persistente, identificando e organizando aqueles menos explorados. Dessa forma, são reveladas áreas com potencial significativo para estudos futuros.

3. Metodologia

Foi utilizado neste trabalho o protocolo desenvolvido por Barbara Kitchenham [13], que enfatiza a precisão metodológica na criação de uma revisão. A revisão se concentra em investigar trabalhos publicados nos últimos 5 anos sobre estruturas de dados aplicadas à memória persistente com endereçamento por *byte*. Esses trabalhos buscam criar uma nova estrutura ou adaptar uma já existente para maximizar o desempenho da PM. Para isso, a pesquisa se baseou em buscar os trabalhos nas principais bases científicas: ACM, IEEE e Scopus. A escolha dessas bases de dados deve-se à sua ampla utilização em pesquisas nas áreas de ciência da computação e engenharia, além de possuírem um extenso acervo de trabalhos. A seguir é apresentada a metodologia utilizada para a pesquisa.

3.1. Pergunta Chave da pesquisa

A principal questão que essa pesquisa pretende responder é:

Qual o estado atual da pesquisa de estruturas de dados para memória persistente?

Assim, procura-se conhecer mais sobre as estruturas de dados em desenvolvimento para memória persistente, realizando uma análise aprofundada de informações relevantes sobre elas. Como resultado, foi possível identificar fatores ainda não explorados ou pouco explorados, como estruturas de dados menos utilizadas e técnicas que podem melhorar o desempenho na memória persistente, visando maximizar o aproveitamento da memória persistente, combinando alto desempenho com persistência.

3.2. Estratégia de Busca

O primeiro passo para o processo de revisão consistiu na escolha das palavras-chave que retornariam estudos sobre o tema desejado. Assim, foi utilizada a seguinte expressão:

```
("data structure") AND ("persistent memory" OR "Storage Class
   Memory" OR "Non-Volatile Memory" OR "Non-Volatile RAM" OR "
   Byte-addressable Persistent RAM" OR "Non-Volatile Main Memory
   " OR "Non-Volatile DIMM" OR "Non-Volatile Byte-addressable
   Memory")
```

Inicialmente, a busca foi realizada por palavras-chave em todo o corpo do texto. No entanto, isso resultou em resultados que não estavam relacionados ao nosso tema, onde as estruturas de dados eram utilizadas em contextos diversos que não se alinhavam ao tema pesquisado. Em virtude disso, a busca foi restringida apenas ao resumo dos artigos. Essa busca foi feita nas seguintes bases, com os respectivos resultados:

- ACM - total de 24 trabalhos
- IEEE - total de 16 trabalhos
- Scopus - total de 146 trabalhos

A pesquisa foi realizada utilizando o sistema de busca integrado à própria base, restringindo os resultados a artigos publicados nos últimos 5 anos. Tal restrição visa entender as tendências atuais de pesquisa, identificando as áreas que estão recebendo maior atenção e esforço.

3.3. Critérios de Inclusão e Exclusão

Para refinar a pesquisa e alcançar os resultados desejados, é necessário estabelecer critérios para a inclusão e exclusão de estudos. A seguir são apresentados ambos critérios.

Tabela 1. Critérios de inclusão e exclusão.

Critério	ID	Descrição
Inclusão	I1	O trabalho deve desenvolver ou adaptar uma estrutura de dados para memória persistente.
	I2	O trabalho deve estar em português ou inglês.
Exclusão	E1	O trabalho apenas analisou o desempenho de uma estrutura de dados.
	E2	O trabalho se concentrou na criação de um arcabouço que utiliza estrutura de dados.
	E3	O trabalho é uma revisão literária.
	E4	O trabalho não foca em memória persistente com endereçamento por *byte*.
	E5	O trabalho não toma cuidado com a persistência dos dados.
	E6	O trabalho deve possuir DOI.

3.3.1. Explicação dos critérios

- **I1 e E1:** Para inclusão no trabalho, é necessário que o trabalho adapte ou desenvolva uma estrutura de dados específica para uso em memória persistente. Trabalhos que apenas testam o desempenho de estruturas de dados, previamente desenvolvidas em outros estudos, na memória persistente, não são de nosso interesse.

- **I2:** Para ser considerado neste trabalho, o trabalho deve estar redigido em inglês ou português, a fim de garantir sua compreensão.
- **E2:** Para ser considerado neste trabalho, é imprescindível que o trabalho tenha como foco principal as estruturas de dados. Trabalhos que se concentram em arcabouços que modificam as estruturas de dados, colocando estas últimas a um papel secundário, não são o objetivo deste artigo.
- **E3:** Artigos que se limitam a revisar trabalhos sobre um determinado tema não são aceitos neste trabalho, uma vez que não contribuem para o desenvolvimento de estruturas de dados para memória persistente.
- **E4:** Para ser considerado neste trabalho, é essencial que o trabalho desenvolva estruturas de dados especificamente para memórias persistentes com endereçamento por *byte*, em vez de memórias persistentes em geral, como SSDs e HDs.
- **E5:** Ao programar para memória persistente é preciso adotar precauções específicas para evitar a perda de dados ou a geração de arquivos corrompidos. No caso de uma falha da máquina, há o risco de perder todos os dados da memória. No exemplo da lista encadeada já visto anteriormente, a perda de um nó pode tornar todos os nós subsequentes inacessíveis, eliminando assim as vantagens do uso da PM. Existem bibliotecas, como a *Persistent Memory Development Kit* (PMDK), que oferecem transações para garantir a persistência dos dados. Caso o estudo em questão não disponha de um sistema que assegure a persistência dos dados, este será descartado.
- **E6:** A necessidade do DOI se deve ao fato de facilitar a localização dos trabalhos, além de permitir a filtragem de *conference reviews* da pesquisa, que não são os trabalhos originais.

3.4. Seleção dos Estudos

A busca inicial realizada em Outubro de 2023 resultou em um total de 186 artigos. No entanto, 15 desses artigos foram removidos por não possuírem um Identificador de Objeto Digital (DOI), restando assim 171 artigos.

Devido ao grande volume de artigos, foi necessário adotar medidas adicionais para restringir a seleção. A primeira etapa foi a remoção de artigos duplicados, utilizando o DOI como referência, o que resultou na exclusão de 40 artigos. Após essa etapa inicial de filtragem, todos os títulos e resumos foram cuidadosamente analisados para refinar ainda mais a seleção. Optou-se por selecionar apenas os estudos que estavam diretamente alinhados ao tema principal da pesquisa. Com isso, o número total de artigos selecionados foi reduzido para 77. Na etapa final, procedeu-se à leitura integral dos artigos restantes. Aqueles que não se enquadravam no tema proposto foram excluídos. Após a conclusão desta etapa, restaram 25 trabalhos para análise. Os passos citados estão ilustrados na Figura 2.

4. Extração de Dados

Subquestões foram elaboradas para aprofundar a compreensão do cenário atual relativo ao tema. Essas perguntas buscam detalhar características importantes que impactam tanto o desempenho quanto o funcionamento das mesmas. A leitura completa dos artigos permitiu responder a essas subquestões. As subquestões formuladas foram:

- **Q1 - Qual a linguagem mais utilizada na implementação das estruturas de dados?**
  A identificação da linguagem de programação mais frequentemente utilizada na

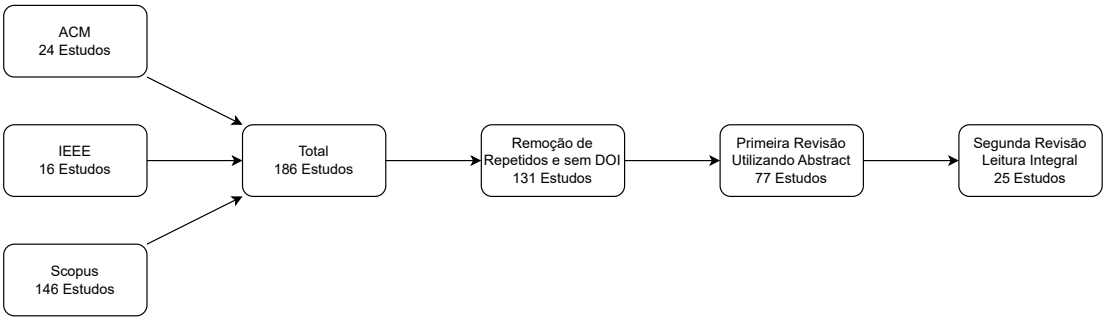

Figura 2. Etapas de Seleção.

construção de estruturas de dados para a memória persistente é fundamental, pois permite determinar qual linguagem é mais adequada para essa finalidade, oferecendo maior controle sobre a PM.

- **Q2 - Quais estruturas de dados são mais utilizadas?**
  Estruturas como árvores e *hash tables* possibilitam buscas significativamente mais rápidas, uma vez que apresentam menor complexidade, quando comparadas a estruturas como filas e listas. A compreensão da estrutura de dados mais empregada possibilita a identificação daquela que otimiza o desempenho da memória persistente.
- **Q3 - Possui código aberto?**
  O código aberto nos oferece a possibilidade de estudar a maneira como a estrutura de dados foi implementada na memória persistente, além de permitir a identificação das bibliotecas utilizadas. Isso facilita a replicação dos experimentos e aumenta a confiabilidade do trabalho.
- **Q4 - Usou sistema simulado para fazer avaliação?**
  Nem todos os estudos dispõem de memória persistente com endereçamento por *byte* para testar as estruturas de dados. Ao utilizar sistemas simulados, pode haver uma perda de precisão nos resultados, mesmo que seja adicionada latência na memória durante a simulação. Quando se simula um disco na DRAM sem a inclusão de latência, o desempenho tende a ser significativamente superior em comparação com qualquer tipo de memória persistente. Portanto, seria relevante determinar a quantidade desses que recorreram a um sistema simulado para avaliar o desempenho.
- **Q5 - Utiliza DRAM como mecanismo para otimização?**
  Para otimizar o acesso à memória persistente, alguns estudos adotam uma abordagem híbrida de armazenamento, estabelecendo um arquivo de cache na memória DRAM. Isso contribui para a redução do número de leituras na PM, aumentando o desempenho. É crucial determinar se foi empregada alguma otimização desse tipo devido às características específicas da memória persistente, como a latência elevada em operações de escrita. Assim, é possível compreender se os estudos utilizam a DRAM para aprimorar algumas dessas características.
- **Q6 - Levou em conta alguma característica específica do dispositivo para otimização?**
  Alguns estudos otimizam as estruturas para minimizar o número de leituras ou escritas realizadas na memória, em comparação com a estrutura convencional implementada na memória DRAM. Ao analisar essas otimizações, torna-se possível examinar as características específicas do dispositivo que devem ser consideradas

durante a programação para esse dispositivo. Essa consideração é imprescindível ao buscar desenvolver uma ED que maximize o desempenho da PM.

5. Resultados

Esta seção apresenta os resultados obtidos a partir das perguntas anteriores, após uma análise detalhada dos artigos selecionados.

5.1. Q1 - Qual a linguagem mais utilizada na implementação das estruturas de dados?

Em nossa análise dos artigos, observamos que 14 deles empregaram C++, uma linguagem de programação conhecida por oferecer um controle melhor sobre a memória. Isso sugere uma preferência potencial por essa linguagem em pesquisas que exigem um gerenciamento de memória mais refinado. No entanto, é importante notar que a linguagem de programação adotada não foi declarada em outros 9 artigos analisados, o que pode indicar uma variedade de abordagens, visto que os mesmos não disponibilizaram o código fonte. Além disso, dois artigos utilizaram C, uma linguagem que, assim como o C++, permite um maior controle da memória. Isso reforça a ideia de que o controle preciso da memória é um fator importante considerado pelos pesquisadores. Os trabalhos foram classificados de acordo com a linguagem de programação utilizada na Tabela 2.

Linguagem	**Trabalhos**
C++	[16, 21, 2, 29, 26, 17, 24, 18, 11, 14, 15, 5, 4, 23]
C	[20, 30]
Não declarado	[6, 9, 19, 22, 27, 28, 10, 25, 12]

Tabela 2. Linguagens utilizadas nos artigos (Q1).

5.2. Q2 - Quais estruturas de dados são mais utilizadas?

Em nossa análise, descobrimos que as estruturas mais empregadas foram o *hash table* e a árvore, cada uma sendo mencionada em 10 artigos. Isso pode indicar uma preferência por essas estruturas devido à sua eficiência em operações comuns como adicionar, remover ou procurar um dado, resultando em um desempenho superior. No entanto, é importante considerar que a escolha da estrutura de dados pode variar dependendo do contexto específico do problema que está sendo resolvido. Além dessas, o grafo foi utilizado em dois artigos, o que pode refletir seu uso comum em aplicações de inteligência artificial. Outras estruturas como lista encadeada, fila e *skip-List* foram mencionadas em um artigo cada, sugerindo que elas podem ser aplicadas em contextos mais específicos ou nichos de pesquisa. Essa variedade de estruturas utilizadas destaca a importância de escolher a estrutura de dados mais adequada para cada aplicação específica. A Tabela 3 agrupa os trabalhos de acordo com a estrutura de dados utilizada.

5.3. Q3 - Possui código aberto?

Durante a análise dos artigos selecionados para este artigo, observou-se que a disponibilização do código-fonte não é uma prática comum. De fato, dos 25 artigos analisados, constatou-se que 15 deles não forneceram acesso direto ao código-fonte. No entanto, é importante destacar que alguns desses artigos compartilharam suas metodologias por meio da apresentação de trechos de códigos na forma de pseudo-código.

Estruturas	Trabalhos
Grafo	[17], [4]
Hash table	[16], [27], [30], [19], [28], [20], [29], [9], [14], [10]
Lista encadeada	[11]
Fila	[6]
Skip-list	[5]
Árvore	[21], [2], [26], [24], [18], [22], [15], [25], [12], [23]

Tabela 3. Artigos por estrutura de dados (Q2).

A disponibilização do código-fonte é uma prática que traz inúmeros benefícios para a comunidade científica. Ela permite uma análise mais aprofundada das técnicas empregadas, oferece a possibilidade de testar e adaptar essas técnicas para outros programas e aumenta a credibilidade do estudo ao permitir a repetição dos experimentos. Para os propósitos deste trabalho, consideramos como código aberto apenas os artigos que incluíram um link direto para o repositório no corpo do texto. Optamos por não realizar buscas adicionais por repositórios em plataformas externas, como o Google, para manter o foco na disposição direta dos autores em compartilhar seus códigos. Na Tabela 4, é possível consultar quais trabalhos possuem código aberto.

Código Fechado	[18, 22, 15, 25, 12, 23, 16, 27, 19, 28, 29, 9, 10, 6, 11]
Código Aberto	[30, 21, 2, 20, 26, 17, 24, 14, 5, 4]

Tabela 4. Artigos com código aberto e código fechado (Q3).

5.4. Q4 - Usou sistema simulado para fazer avaliação?

Entre os estudos selecionados para esta análise, observou-se que 15 optaram por testar o desempenho das estruturas de dados diretamente na memória persistente, utilizando a tecnologia Intel Optane. Dos 10 estudos restantes, dois recorreram ao programa Quartz [20, 23] para simular o ambiente de teste e um utilizou o Gem 5 [30]. Essas informações estão detalhadas na Tabela 5.

Alguns estudos adotaram uma abordagem diferente, optando por gerar um disco virtual na DRAM. Para isso, utilizaram uma ferramenta disponível no próprio sistema operacional Linux, sem adicionar qualquer latência à memória. No entanto, é importante ressaltar que essa abordagem tem suas limitações. Ao criar um disco na memória DRAM sem considerar a latência, torna-se inviável comparar o desempenho com outros testes realizados na memória persistente real. Isso se deve ao fato de que o desempenho da DRAM é superior. Portanto, esse tipo de teste permite apenas a verificação do funcionamento da estrutura de dados, e não uma comparação justa do desempenho.

5.5. Q5 - Utiliza DRAM como mecanismo para otimização?

A memória DRAM, com sua versatilidade e velocidade, oferece uma gama de possibilidades para otimização de desempenho. Uma estratégia comum é carregar dados completos na DRAM, permitindo um acesso mais rápido em comparação com a memória

Não simulam	[21], [26], [24], [14], [5], [4], [15], [16], [19], [28], [29], [9], [10], [6], [11]
Não nomeados	[27], [2], [17], [18], [22], [25], [12]
Quartz	[20], [23]
Gen 5	[30]

Tabela 5. Artigos que utilizaram simulação (Q4).

persistente. Alternativamente, pode-se optar por armazenar apenas as partes mais frequentemente acessadas ou um índice da estrutura de dados na DRAM. Essas abordagens, embora distintas, compartilham o objetivo comum de maximizar a eficiência e o desempenho do sistema.

Observou-se que a maioria dos artigos selecionados para este trabalho empregou a DRAM como uma estratégia para otimizar o desempenho. Eles utilizaram parte das estruturas de dados na DRAM como cache, acelerando assim o processo de leitura. Dos artigos analisados, 19 optaram por essa abordagem, enquanto 6 decidiram não utilizar a DRAM, conforme demonstrado na Tabela 6. A maioria dos estudos que empregaram árvores e criaram *caches* optou por uma estratégia específica: armazenar os dados na memória persistente, enquanto a estrutura da árvore era gerada na memória DRAM. Essa abordagem tem uma vantagem significativa, pois elimina a necessidade de acessar a memória persistente para a busca de um nó. Isso acelera a maioria dos processos, já que o acesso à memória persistente se torna necessário apenas para a remoção ou adição de um valor.

Não Utilizada	[27], [21], [9], [11], [14], [23]
Utiliza	[30], [2], [20], [26], [17], [24], [5], [4], [18], [22], [15], [25], [12], [16], [19], [28], [29], [10], [6]

Tabela 6. Artigos que utilizaram DRAM para otimização (Q5).

5.6. Q6 - Levaram em conta alguma característica específica do dispositivo para otimização?

A maioria dos estudos analisados neste artigo emprega algum tipo de otimização com o objetivo de aprimorar o desempenho das estruturas de dados, minimizando a quantidade de operações de leitura ou escrita. Muitos desses estudos focam especificamente na redução do número de operações de escrita, uma vez que a performance da memória persistente tende a ser inferior durante essas operações, apresentando uma latência superior à das operações de leitura.

Dos 25 estudos analisados, apenas 7 não implementaram otimizações específicas nas estruturas de dados, limitando-se a adaptá-las para uso com memória persistente. No entanto, é importante ressaltar que esses sete artigos propuseram a criação de uma nova estrutura de dados. Isso sugere que a inovação e a adaptação são considerações importantes na pesquisa e desenvolvimento de estruturas de dados para memória persistente. Os trabalhos estão agrupados na Tabela 7.

É importante ressaltar que, além dos métodos empregados nos estudos em questão, há uma gama de outros não explorados. Entre eles, destacamos os padrões de acesso, uma

estratégia potencialmente benéfica para certos tipos de memória que se favorecem de acessos sequenciais. Também existe a prática de evitar escritas repetidas no mesmo local, que pode ser uma medida eficaz para prevenir o desgaste desproporcional da memória. Essas abordagens, embora não tenham sido objeto de nossa análise, merecem ser consideradas em futuras investigações. A inclusão dessas estratégias pode enriquecer o escopo do trabalho e fornecer informações adicionais sobre a otimização do uso da memória.

Implementaram Otimizações	[18], [22], [15], [23], [27], [19], [28], [29], [10], [6], [30], [2], [26], [17], [24], [14], [5], [4]
Não Implementaram	[16], [21], [20], [9], [11], [25], [12]

Tabela 7. Artigos que implentaram otimizações para a PM (Q6).

6. Discussão

Uma análise dos estudos revela um padrão recorrente em suas abordagens. A linguagem C++ é frequentemente escolhida devido ao controle mais granular que oferece ao programador sobre a memória. Além disso, a DRAM é comumente utilizada como uma estratégia de otimização, mantendo a estrutura de dados nela para evitar o acesso à PM, assim, melhorar o desempenho. As implementações de otimizações, como a redução da quantidade de escritas na memória persistente, são outra característica comum nos trabalhos. Essas práticas, observadas na maioria dos estudos analisados, refletem o interesse crescente em aprimorar o desempenho da memória persistente em diversos cenários.

Quando se trata da estrutura de dados mais utilizada, as *hash tables* e as árvores se destacam pelo seu desempenho superior em operações de busca e inserção de dados. Foram apresentadas várias versões dessas estruturas de dados, algumas focando no paralelismo, enquanto outras buscam trazer persistência sem comprometer o desempenho. Essas tendências evidenciam a busca contínua por soluções que equilibram eficiência e desempenho na manipulação de dados em memória persistente. Um ponto importante ao comentar sobre os estudos que não simularam seus testes, a maioria utilizou a memória persistente da Intel, Intel Optane 3D, para executar os testes.

Infelizmente a maioria dos trabalhos não disponibiliza os repositórios de códigos fonte em seus artigos. Esta prática limita a replicação dos testes e a adaptação potencial dessas estruturas em outras pesquisas. Acreditamos firmemente que a disponibilidade do código promove o crescimento coletivo da comunidade científica, permitindo a todos contribuir para o desenvolvimento e aprimoramento das tecnologias existentes.

7. Conclusão

Neste artigo, observou-se uma tendência recorrente entre as pesquisas, onde a maioria concentra-se em objetivos similares: aprimorar o desempenho das estruturas de dados ao minimizar o número de gravações na memória persistente, empregando estruturas de dados que priorizam a eficiência, como *hash table* e árvores. Constatamos que a quantidade de pesquisas dedicadas ao estudo de grafos, uma estrutura de dados muito utilizada atualmente, é comparativamente menor do que o esperado, um achado que nos instigou a investigar mais a fundo essa área. Com o fim do suporte para os dispositivos Optane DC da Intel, esperamos que novos trabalhos comecem a focar na padrão Compute Express Link (CXL) [3], que também provê suporte para PM. No entanto, ainda não observarmos esta tendência refletida nos artigos pesquisados.

Agradecimentos. Os autores agradecem à Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), processos nº 2018/15519-5, 2019/26702-8, 2023/04971-2 e 2023/04969-8 pelo apoio a este trabalho.

Referências

[1] BALDASSIN, A., BARRETO, J., CASTRO, D., AND ROMANO, P. Persistent memory: A survey of programming support and implementations. *ACM Comput. Surv. 54*, 7 (July 2021).

[2] COHEN, N., AKSUN, D. T., AVNI, H., AND LARUS, J. R. Fine-grain checkpointing with in-cache-line logging. In *Proceedings of the 24th ASPLOS* (2019), p. 441–454.

[3] DESNOYERS, P., ADAMS, I., ESTRO, T., GANDHI, A., KUENNING, G., MESNIER, M., WALDSPURGER, C., WILDANI, A., AND ZADOK, E. Persistent memory research in the post-Optane era. In *Proceedings of the 1st DIMES* (2023), p. 23–30.

[4] DHULIPALA, L., MCGUFFEY, C., KANG, H., GU, Y., BLELLOCH, G. E., GIBBONS, P. B., AND SHUN, J. Sage: Parallel semi-asymmetric graph algorithms for NVRAMs. *Proceedings of the VLDB Endowment 13*, 9 (2020), 1598 – 1613.

[5] DUAN, Z., YAO, J., LIU, H., LIAO, X., JIN, H., AND ZHANG, Y. Revisiting log-structured merging for kv stores in hybrid memory systems. In *Proceedings of the 28th ASPLOS* (2023), p. 674–687.

[6] FRIEDMAN, M., HERLIHY, M., MARATHE, V., AND PETRANK, E. A persistent lock-free queue for non-volatile memory. In *Proceedings of the 23rd PPoPP* (2018), p. 28–40.

[7] GRAY, J., AND REUTER, A. *Transaction Processing: Concepts and Techniques*, 1st ed. Morgan Kaufmann, 1992.

[8] HENNESSY, J. L., AND PATTERSON, D. A. *Computer Architecture: A Quantitative Approach*, 6th ed. Morgan Kaufmann, Nov. 2017.

[9] HU, J., CHEN, J., ZHU, Y., YANG, Q., PENG, Z., AND YU, Y. Parallel multi-split extendible hashing for persistent memory. In *Proceedings of the 50th ICPP* (2021).

[10] HUANG, K., YAN, Y., AND HUANG, L. Revisiting persistent hash table design for commercial non-volatile memory. *Proceedings of the 2020 Design, Automation and Test in Europe Conference and Exhibition, DATE 2020* (2020), 708 – 713.

[11] IZADPANAH, R., PETERSON, C., SOLIHIN, Y., AND DECHEV, D. Petra: Persistent transactional non-blocking linked data structures. *ACM Transactions on Architecture and Code Optimization 18*, 2 (2021).

[12] JAMIL, S., KHAN, A., BURASTALLER, B., AND KIM, Y. Towards scalable manycore-aware persistent b+- trees for efficient indexing in cloud environments. In *2021 ACSOS-C* (2021), pp. 44–49.

[13] KITCHENHAM, B. Procedures for performing systematic reviews. *Keele, UK, Keele Univ. 33* (08 2004).

[14] LAMAR, K., PETERSON, C., DECHEV, D., PEARCE, R., IWABUCHI, K., AND PIRKELBAUER, P. PMap: A non-volatile lock-free hash map with open addressing. In *2021 NVMSA* (2021), pp. 1–7.

[15] LAVINSKY, B., AND ZHANG, X. PM-Rtree: A highly-efficient crash-consistent r-tree for persistent memory. *Proceedings of the 34th International Conference on Scientific and Statistical Database Management* (2022).

[16] LI, Y., ZENG, L., CHEN, G., GU, C., LUO, F., DING, W., SHI, Z., AND FUENTES, J. A multi-hashing index for hybrid dram-nvm memory systems. *Journal of Systems Architecture 128* (2022).

[17] LIM, S., COY, T., LU, Z., REN, B., AND ZHANG, X. NVGraph: Enforcing crash consistency of evolving network analytics in nvmm systems. *IEEE Transactions on Parallel and Distributed Systems 31*, 6 (2020), 1255 – 1269.

[18] NGUYEN, B., TAN, H., DAVIS, K., AND ZHANG, X. Persistent octrees for parallel mesh refinement through non-volatile byte-addressable memory. *IEEE Transactions on Parallel and Distributed Systems 30*, 3 (2019), 677 – 691.

[19] OH, H., CHO, B., KIM, C., PARK, H., AND SEO, J. Anifilter: Parallel and failure-atomic cuckoo filter for non-volatile memories. *Proceedings of the 15th European Conference on Computer Systems, EuroSys 2020* (2020).

[20] PAN, W., XIE, T., AND SONG, X. Hart: A concurrent hash-assisted radix tree for dram-pm hybrid memory systems. *Proceedings - 2019 IEEE 33rd International Parallel and Distributed Processing Symposium, IPDPS 2019* (2019), 921 – 931.

[21] SRIVASTAVA, A., AND BROWN, T. Elimination (a,b)-trees with fast, durable updates. *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP* (2022), 416 – 430.

[22] WANG, C., WEI, Q., WU, L., WANG, S., CHEN, C., XIAO, X., YANG, J., XUE, M., AND YANG, Y. Persisting rb-tree into nvm in a consistency perspective. *ACM Transactions on Storage 14*, 1 (2018).

[23] WANG, H., LI, Z., ZHANG, X., ZHAO, X., AND JIANG, S. Wobtree: a write-optimized b+-tree for non-volatile memory. *Frontiers of Computer Science 15*, 5 (2021).

[24] YAO, Z., ZHANG, J., AND FENG, J. NV-QALSH: An NVM-optimized implementation of query-aware locality-sensitive hashing. *Lecture Notes in Computer Science 12924 LNCS* (2021), 58 – 69.

[25] YU, S., XIAO, N., DENG, M., XING, Y., LIU, F., AND CHEN, W. Rio: Fast b+-tree based on remote accessible non-volatile memory. In *2017 IEEE ISPA/IUCC* (2017), pp. 494–496.

[26] ZHANG, B., ZHENG, S., QI, Z., AND HUANG, L. Nbtree: a lock-free pm-friendly persistent b+-tree for eadr-enabled pm systems. *Contemporary Mathematics 15*, 6 (2022), 1187 – 1200.

[27] ZHANG, X., FENG, D., HUA, Y., CHEN, J., AND FU, M. A write-efficient and consistent hashing scheme for non-volatile memory. *ACM International Conference Proceeding Series* (2018).

[28] ZHONG, C., CHALLA, P., ZHAO, X., AND JIANG, S. Buffered hash table: Leveraging dram to enhance hash indexes in the persistent memory. *Proceedings - 2022 IEEE 11th Non-Volatile Memory Systems and Applications Symposium, NVMSA 2022* (2022), 8 – 13.

[29] ZHU, J., HUANG, K., ZOU, X., HUANG, C., XU, N., AND FANG, L. Hdnh: A read-efficient and write-optimized hashing scheme for hybrid dram-nvm memory. *ACM International Conference Proceeding Series* (2021).

[30] ZUO, P., AND HUA, Y. A write-friendly and cache-optimized hashing scheme for non-volatile memory systems. *IEEE Transactions on Parallel and Distributed Systems 29*, 5 (2018), 985 – 996.