Avaliação de Caches em Dispositivos de Armazenamento Secundário com SSDs

Leonardo A. dos Santos, Carlos A. Maziero e Luis Carlos E. de Bona
Programa de Pós-Graduação em Informática
Universidade Federal do Paraná
Curitiba, Paraná, Brasil
E-mail:{lasantos,maziero,bona}@inf.ufpr.br

***Resumo*—Recentemente, os discos de estado sólido (SSDs – *Solid State Disks*) elevaram muito o desempenho no acesso ao armazenamento secundário. Contudo, seu custo elevado e baixa capacidade inviabilizam a substituição integral dos discos rígidos (HDDs – *Hard Disk Drives*) por SSDs a curto prazo, sobretudo em instalações de maior porte. Por outro lado, é possível aliar o desempenho dos SSDs à capacidade e baixo custo dos HDDs, usando SSDs como cache dos HDDs para os dados mais acessados, de forma transparente às aplicações. Essa abordagem é usada nos discos híbridos, que são HDDs com um pequeno cache interno em estado sólido, geralmente gerenciado pelo *firmware* do próprio disco. Também é possível usar SSDs independentes como cache de HDDs subjacentes, com o gerenciamento feito pelo sistema operacional. O núcleo Linux oferece dois subsistemas de gerenciamento de caches em SSD, o *DMCache* e o *BCache*, que usam abordagens e algoritmos distintos. Este trabalho visa avaliar estes dois subsistemas em diversas configurações de SSDs e HDDs, sob diversas cargas de trabalho, com o objetivo de compreender seu funcionamento, identificar problemas, propor melhorias e definir diretrizes para a configuração de tais subsistemas em ambientes computacionais de médio/grande porte.**

I. Introdução

A popularidade dos serviços de Internet, em especial os mecanismos de busca, comércio eletrônico e armazenamento em nuvem, tem provocado a mudança de foco do processamento para a comunicação e armazenamento da informação, aumentado a demanda de capacidade e desempenho por parte das tecnologias de armazenamento secundário, tanto na computação pessoal quanto nos *datacenters*. A tecnologia mais popular de armazenamento secundário ainda é o disco magnético (HDD – *Hard Disk Drive*), que alia grande capacidade de armazenamento e baixo custo por MB. Contudo, o HDD tem tempos de acesso elevados, devido às suas características mecânicas (latência rotacional e de movimento da cabeça de leitura). Essas latências impactam sobretudo os acessos aleatórios, levando a um baixo desempenho.

Recentemente, os discos de estado sólido (SSDs – *Solid State Disks*) elevaram muito o desempenho no acesso ao armazenamento secundário. Contudo, seu custo elevado e baixa capacidade inviabilizam a substituição integral dos discos magnéticos por SSDs a curto prazo, sobretudo em instalações de maior porte. Por outro lado, é possível aliar o desempenho dos SSDs à capacidade e baixo custo dos HDDs, usando SSDs como *cache* dos HDDs para os dados mais acessados, de forma transparente às aplicações. Essa abordagem é usada por exemplo nos discos híbridos, que são HDDs com um pequeno cache interno em estado sólido, gerenciado pelo *firmware* do próprio disco. Também é possível usar SSDs independentes como cache de HDDs subjacentes, com o gerenciamento feito pelo sistema operacional.

O dispositivo de cache é mais rápido, mas também mais caro e menor que o dispositivo de origem dos dados. Por isso, os dados do dispositivo original não cabem todos no cache, o que leva à necessidade de *algoritmos de substituição* para manter no cache os dados mais relevantes a cada momento. O núcleo Linux oferece dois subsistemas de gerenciamento de cache em SSD, o *DMCache* [1] e o *BCache* [2], que usam abordagens e algoritmos distintos.

Outra forma usual de aumentar o desempenho de sistemas de armazenamento secundário com HDDs consiste em combinar diversos discos em arranjos RAID (*Redundant Array of Independent Disks*) [3]. Por permitir acessos paralelos aos discos físicos, um sistema RAID configurado adequadamente oferece ganhos significativos no desempenho de acesso aos dados. É importante observar que sistemas RAID e caches SSD podem ser usados em conjunto.

Este trabalho visa estudar o comportamento dos subsistemas de cache em SSD oferecidos pelo núcleo Linux, considerando diversos arranjos de discos SSD e HDD, com várias cargas de trabalho. O objetivo principal é compreender o funcionamento desses subsistemas, mas também identificar problemas, propor melhorias e definir diretrizes para a configuração de arranjos SSD/HDD em ambientes computacionais de maior porte.

O texto está organizado da seguinte forma: a Seção II apresenta os dispositivos de armazenamento secundário considerados neste trabalho; na Seção III são apresentados algoritmos de substituição de cache usuais e os mecanismos de gestão de caches SSD disponíveis no núcleo Linux; a Seção IV descreve o *setup* experimental e a metodologia utilizada; a Seção V apresenta e discute os resultados obtidos; por fim, a Seção VI discute os trabalhos relacionados e a Seção VII conclui o artigo.

II. Estruturas de Armazenamento Secundário

O armazenamento secundário permite a manutenção não-volátil de dados, que são preservados mesmo com o sistema desligado. Esse armazenamento pode ser *online*, quando os dispositivos estão diretamente acessíveis (como HDDs ou SSDs), ou *offline*, quando há necessidade de intervenção externa para acesso (como discos óticos e fitas magnéticas). Esta Seção apresenta os principais dispositivos de armazenamento

secundário *online*, bem como as possibilidades de arranjos de discos RAID.

A. Discos rígidos magnéticos (HDD)

Os discos rígidos magnéticos (HDDs) são hoje a forma mais usual de armazenamento secundário *online* de dados. Fisicamente, um HDD possui um ou mais discos metálicos circulares (*platters*), cuja composição magnética permite que dados sejam gravados em sua superfície. Cada superfície magnética é lida/escrita por uma cabeça de leitura; as cabeças de leitura estão conectadas entre si em um braço motorizado que as posiciona conforme o local do disco a ser acessado. O HDD possui também um motor responsável por girar em alta velocidade os discos magnéticos.

O desempenho de um disco rígido depende diretamente de sua *taxa de transferência* e de seu *tempo de acesso*. A taxa de transferência está relacionada ao posicionamento dos dados no disco (trilhas mais internas ou mais externas), à velocidade de rotação e ao barramento de dados. Por sua vez, o tempo de acesso está relacionado ao tempo necessário para posicionar o braço de leitura no cilindro desejado (*seek time*, tipicamente entre 4 e 10 ms) e para girar o disco de modo que o dado desejado esteja sob a cabeça de leitura (*rotational latency*, tipicamente entre 3 e 7 ms). A redução do tempo de acesso é um dos principais objetivos dos escalonadores de disco.

B. Discos de estado sólido (SSD)

Os discos de estado sólido são construído com memória semicondutora não-volátil. Por não possuir partes mecânicas, seus tempos de acesso e consumo são bem inferiores aos HDDs. A maioria dos SSDs em uso atualmente são construídos com tecnologia NAND *flash memory*. As células de memória NAND armazenam elétrons em um capacitor por tempo indefinido sem o uso de energia. A escrita ou exclusão de dados nesses chips é feita adicionando ou removendo elétrons da célula de memória, através de pulsos de alta voltagem. A leitura dos bits é realizada através de um conversor analógico-digital, com a injeção de uma tensão de *bias* nas células lidas. Diferentes tipos de memória utilizam limites distintos para determinar o valor de uma célula de memória [4].

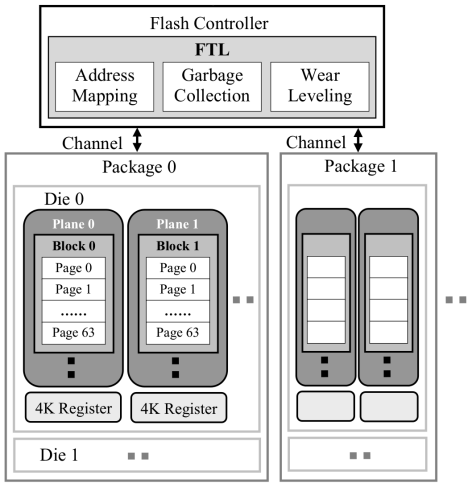

Figura 1. Arquitetura dos SSDs (adaptada de [5]).

Como mostra a Figura 1, um SSD é composto por células de memória *flash* agrupadas em páginas, sendo estas agrupadas em blocos. Devido à estrutura da memória, a unidade básica de leitura/escrita é a página. Além disso, uma página só pode ser escrita se tiver sido previamente apagada (*erased*). O apagamento de células é feito por blocos inteiros, através de pulsos de alta tensão. Por isso, caso uma página deva ser escrita, mas não existam páginas apagadas disponíveis, um apagamento de bloco deve ser previamente efetuado. Um mecanismo interno de *garbage collection* periodicamente reorganiza as páginas nos blocos e apaga blocos sem uso, para manter a oferta de páginas apagadas.

Outro aspecto importante das memórias *flash* é que estas têm vida limitada: após uma certa quantidade de operações, as células perdem a capacidade de armazenamento. Por isso, o controlador SSD deve implementar mecanismos de *wear leveling* para distribuir de modo uniforme o desgaste de uso entre as células e assim prolongar a vida útil do dispositivo. Em conjunto, as técnicas de *garbage collection* e *wear leveling* produzem uma quantidade de escritas superior à gerada pelas aplicações, causando o fenômeno de *write amplification*, o que penaliza os tempos de escrita [6].

C. Arranjos RAID

A tecnologia RAID (*Redundant Array of Independent Disks*, [3]) consiste em agregar vários discos físicos, por meio de uma placa controladora específica, e apresentá-los ao sistema como um único disco lógico. Fazendo uso de paralelismo e redundância, os arranjos RAID permitem ganhos consideráveis na velocidade de acesso aos dados e também podem prover tolerância a falhas nos discos físicos.

Os arranjos RAID são classificados em níveis (0, 1, etc.), cada um com suas características próprias de paralelismo e redundância [3]. Neste trabalho serão considerados os níveis RAID0 e RAID10, por serem as configurações com melhor desempenho de leitura/escrita de dados; aspectos de tolerância a falhas não são considerados neste trabalho.

Na configuração RAID0, cada disco físico é dividido em faixas (*stripping*) de k blocos cada. As faixas dos discos físicos são agrupadas em uma sequência *round-robin* para construir o disco lógico. Esta configuração permite paralelizar as leituras e escritas de dados que estiverem em discos físicos distintos, mas não oferece nenhuma redundância. Por sua vez, na configuração RAID1 os dados são replicados (espelhados) entre os discos físicos, trazendo tolerância a falhas ao sistema. As leituras ainda podem ser paralelizadas entre os discos físicos, mas nas escritas todos os discos devem ser atualizados, o que limita o ganho de desempenho. Finalmente, a configuração RAID10 cria um arranjo RAID0 (*stripping*) a partir de dois ou mais arranjos RAID1 (*mirroring*), trazendo ganhos de desempenho e redundância ao sistema.

III. Gestão de Caches em SSD

Conforme discutido na Seção I, os SSDs têm custo elevado e baixa capacidade, o que inviabiliza a substituição de HDDs por SSDs em sistemas de maior porte. Contudo, SSDs podem ser usados como *cache* dos HDDs para os dados mais acessados, de forma transparente às aplicações. A gestão do conetúdo do cache pode ser feita pelo *firmware* do disco rígido, no caso

dos discos híbridos, ou pelo sistema operacional, quando forem usados SSDs e HDDs independentes.

Como o cache em SSD é bem menor que o espaço de armazenamento em HDD subjacente, são necessários *algoritmos de substituição* para manter no cache os dados mais relevantes a cada momento e maximizar a taxa de requisições que podem ser atendidos pelos dados presentes no cache (taxa de acertos ou *hit ratio*). Quando um dado não se encontra no cache (erro ou *cache miss*), ele deve ser lido do HDD subjacente, prejudicando o desempenho do sistema. Esta Seção apresenta alguns algoritmos clássicos de substituição de cache; em seguida, descreve os subsistemas de gerenciamento de cache SSD *DMCache* [1] e *BCache* [2], presentes no núcleo Linux.

A. Algoritmos clássicos

O algoritmo ótimo (OPT) para a gestão de cache é capaz de antever a sequência de blocos do armazenamento secundário que serão acessados pelo sistema operacional no futuro; dessa forma, é capaz de identificar os blocos do cache menos requisitados e substitui-los, gerando o mais alto *hit ratio* possível. Como normalmente essa sequência de requisições não pode ser prevista, esse algoritmo não é implementável. Contudo, seus resultados servem como base de comparação para algoritmos reais.

Uma aproximação implementável do algoritmo OPT é o algoritmo *Least Recently Used* (LRU). Inspirado pelo princípio da localidade de referências, o LRU considera que blocos recentemente acessados têm maior chance de serem novamente acessados em um futuro próximo. Apesar da simplicidade, o custo de implementação do LRU pode ser elevado, pois a cada acesso aos dados é necessário atualizar uma fila de blocos acessados, ordenada por data de último acesso. A manutenção dessa fila pode ter um custo significativo.

O algoritmo *Multi-Queue* (MQ) [7] foi desenvolvido para caches de segundo nível (SSD). É um algoritmo de característica local, pois decide que bloco será substituído sem ter informações do primeiro nível de cache (em RAM). Assim, não são necessárias modificações na estrutura dos caches de primeiro nível (do sistema de arquivos, por exemplo). O princípio deste algoritmo é manter os blocos em cache por diferentes períodos de tempo, de acordo com sua frequência de acessos. Para tal, o MQ mantém várias filas LRU (Q_0 a Q_m, com m configurável). Blocos mantidos em uma fila Q_j possuem mais tempo em cache que blocos mantidos em filas $Q_{i<j}$. Além disso, blocos removidos do cache têm as suas informações de frequência armazenadas em um *history buffer*, caso sejam novamente acessados.

A gestão das filas LRU é feita da seguinte forma: para cada bloco b é mantida sua frequência de acessos $f(b)$; um bloco b com frequência de acessos $f(b)$ é colocado no final da fila Q_k onde $k = \log_2 f(b)$. A remoção de blocos do cache inicia na fila Q_0; caso esteja vazia, busca-se em Q_1 e assim por diante, sempre removendo o bloco LRU. Os blocos são mantidos em cada fila Q_i durante um tempo definido pelo parâmetro *expireTime*; ao expirar esse valor, o bloco é movido para a fila inferior Q_{i-1}. O algoritmo MQ apresenta uma complexidade $\mathcal{O}(1)$, sendo portanto mais rápido e escalável que outros algoritmos de substituição de cache [7].

Além dos algoritmos OPT, LRU e MQ, os algoritmos LIRS [8] e ARC [9] figuram entre os algoritmos de substituição modernos que consideram não só a recência, mas também a frequência de acessos. Desses algoritmos, apenas o LRU e MQ são utilizados para substituição de cache de segundo nível nos sistemas avaliados neste trabalho.

B. BCache

O mecanismo BCache [2] permite usar dispositivos SSD como cache de discos HDD. Ele opera na camada de blocos, sendo portanto independente do sistema de arquivos (*filesystem agnostic*) e transparente. O mecanismo BCache oferece às camadas superiores do sistema operacional um disco virtual que abstrai a associação SSD+HDD e seus algoritmos internos.

Como os HDDs apresentam boa vazão em acessos sequenciais e baixa vazão em acessos aleatórios, devido a fatores como o *seek time* [10], o BCache normalmente não faz cache de operações sequenciais, apenas de leituras/escritas aleatórias. A distinção entre operações sequenciais e aleatórias é feita por limites ajustáveis de tamanho de escritas e leituras contíguas: por padrão, requisições de acesso com 4 MB ou mais são consideradas sequenciais. Há também parâmetros para a detecção de I/O aleatório. Por exemplo, o BCache mantém uma média variável das operações de I/O das tarefas e não realiza cache para operações posteriores de uma tarefa caso sua média ultrapasse o limiar estabelecido; deste modo, *backups* e operações em grandes arquivos não afetam o cache.

A unidade básica de alocação de dados do BCache é o *bucket* ("balde"), cujo tamanho típico é 1 MB. A cada *bucket* é associada uma *prioridade*, incrementada a cada *hit* nos dados do *bucket*; essa prioridade serve como base para implementar uma política LRU. As prioridades de todos os *buckets* são periodicamente decrementadas. *Buckets* com prioridade zero são colocados em uma lista de *buckets* livres. Cada *bucket* também possui um atributo *geração*, que serve para agilizar a invalidação e reuso dos *buckets* (*garbage collection*). O conteúdo de cada *bucket* sempre é preenchido sequencialmente, até completá-lo. Além disso, o tamanho de cada *bucket* corresponde ao tamanho dos blocos de apagamento (*erasure*) do SSD. Dessa forma, são evitados problemas de amplificação de escritas no SSD.

C. DMCache

Assim como o BCache, o DMCache [11] também visa criar volumes híbridos com SSDs e HDDs. O DMCache se diferencia do BCache em: a) ele demanda ao menos três dispositivos (ou partições), para *dados*, *caching* e *metadados*; b) sua implementação usa a estrutura de mapeamento *device-mapper* [12], tornando sua implementação mais simples e menos intrusiva no núcleo; c) as políticas de cache são implementados separadamente, como módulos configuráveis.

O DMCache possui três modos de operação que determinam como ocorre a sincronia entre os SSDs e os HDDs; estão disponíveis os modos *write-back* (padrão), *write-through* e *write-around* (ou *pass-through*). O modo *write-back* escreve os blocos no *cache* e os descarrega no dispositivo subjacente obedecendo as regras da política de cache ou quando os blocos estão "sujos" (*dirty*). O modo *write-through* escreve em ambos os dispositivos, confirmando cada escrita apenas

quando completada no dispositivo subjacente. O modo *write-around* usa apenas o dispositivo subjacente; é útil em casos de inconsistência nos dados e inserção de novos dispositivos de cache.

As políticas de cache definem como os dados migram entre os dispositivos SSD e HDD; estão disponíveis as políticas *multiqueue* (padrão) e *cleaner*. A política *multiqueue* possui parâmetros semelhantes ao BCache, como, por exemplo, limites de identificação de acesso sequencial e aleatório (que no DM-Cache são ajustáveis dinamicamente). O DMCache implementa uma adaptação do algoritmo MQ (Seção III-A), enquanto o BCache usa o algoritmo LRU. A política *cleaner* apenas escreve todos os blocos sujos no dispositivo subjacente e cancela as escritas futuras no cache, o que é útil em casos de alterações nos dispositivos, como remoção ou troca de SSDs [1].

IV. Metodologia de avaliação

Este estudo avalia o comportamento dos mecanismos BCache e DMCache em diversas situações de configuração e carga de trabalho. Esta Seção apresenta o ambiente computacional, as ferramentas e a metodologia usada para esta avaliação.

Os experimentos foram efetuados em um computador com processador Intel Xeon E5620 2.40 GHz e 12GB de RAM. Foram usados HDDs Hitachi HUA723030ALA640 de 2TB, 7.200 RPM, 64 MB de cache interna, interface SATA 6 Gb/s, taxa de transferência sustentada de 157 MB/s e *random seek time* de 8.2 ms [13]. Como cache foram usados SSDs Intel 320 Series SSDSA2BW160G3 de 160 GB, interface SATA 3.0, 3 Gb/s, 39.000 IOPS de *random read* e 21.000 IOPS de *random write* para alcance de 8 GB e 600 IOPS para alcance de 100% do dispositivo [14]. Os arranjos RAID foram construídos usando o utilitário *MDAdm* do Linux. As versões de software utilizadas foram: Kernel 3.17.3 (*commit* 7623e24), IOstat/SysStat 10.0.5 e Fio 2.0.8.

Para os experimentos foi usada a ferramenta *Fio* [15], muito usada em testes de *stress* em dispositivos de armazenamento. Ela possui suporte a vários *engines* de I/O (*sync*, *mmap*, *libaio*, etc), taxas de I/O e *jobs* configuráveis, entre outras opções. A configuração usada foi:

- *IOengine: sync*; gerar requisições síncronas, garantindo a ordem das requisições;
- *Direct: true*; operações *open* são realizadas com a *flag O_DIRECT*, minimizando o *caching* em RAM.
- *Size:* $2\times$ *ssd_cache_size*; a quantidade de dados lidos/escritos durante o *workload*; precisa ser maior que o tamanho do cache, para evitar 100% de *hits* [16].
- *Loops: 10*; número de execuções de cada teste sobre o mesmo conjunto de dados. Acessos repetidos ao mesmo conjunto de dados simulam mais fielmente cargas reais.
- *RandRepeat: true*; permite que blocos sejam acessados mais de uma vez em *workloads* aleatórios (o que é necessário para testes de cache).
- *BS: 64 KB, 512 KB ou 4 MB*; tamanho das requisições de I/O nos testes (um tamanho em cada teste).

Neste texto adotamos o termo *dispositivo de origem* para o dispositivo subjacente de onde os dados se originam (os HDDs); para o cache SSD é usado o termo *dispositivo de cache* e para o arranjo que envolve o dispositivo de cache e o dispositivo de origem é usado o termo *arranjo de cache*.

Os resultados estão divididos entre operações de leitura e escrita com padrão de acesso aleatório[1]. Em todos os casos de teste, o arranjo de cache teve seu tamanho ajustado em 30 GB, correspondente a 20% do dispositivo de origem, com tamanho ajustado em 150 GB. A quantidade de memória RAM foi ajustada com 20% do tamanho do cache, ou seja, 6 GB, o que facilita a aferição dos resultados com a mínima interferência dos caches em memória. A proporção de 20% foi escolhida por ser uma relação de uso comum em sistemas reais para caches. O tamanho dos dispositivos físicos foi ajustado por simples particionamento. Nos casos de arranjos com RAID, cada membro do arranjo RAID recebeu uma redução proporcional.

Para os testes foram selecionados 15 arranjos distintos envolvendo HDDs e SSDs, RAID0 e RAID10, DMCache e BCache, escolhidos por serem os mais representativos de situações distintas. A Tabela I apresenta a composição dos arranjos escolhidos. O nome dos arranjos reflete sua estrutura: por exemplo, o nome "*4h0*" indica um arranjo com 4 HDDs em configuração RAID0, sem mecanismo de cache. Já o nome "*bcache-4s0-4h10*" indica um arranjo de 4 SSDs em RAID0 e 4 HDDs em RAID10, usando o mecanismo *BCache*. HDDs e SSDs isolados (*1h* e *1s*) foram avaliados como referências para os demais resultados. Os níveis RAID0 e RAID10 foram escolhidos por sua popularidade, desempenho e redundância.

Tabela I. Composição dos arranjos utilizados nos testes.

Arranjo de cache	# dispositivos SSDs	# dispositivos HDDs	Nível RAID SSDs	Nível RAID HDDs	Mecanismo de cache
1h	-	1	-	-	-
1s	1	-	-	-	-
4h0	-	4	-	raid0	-
4s0	4	-	raid0	-	-
4h10	-	4	-	raid10	-
dmcache-1s-1h	1	1	-	-	dmcache
bcache-1s-1h	1	1	-	-	bcache
dmcache-1s-4h0	1	4	-	raid0	dmcache
bcache-1s-4h0	1	4	-	raid0	bcache
dmcache-1s-4h10	1	4	-	raid10	dmcache
bcache-1s-4h10	1	4	-	raid10	bcache
dmcache-4s0–4h0	4	4	raid0	raid0	dmcache
bcache-4s0-4h0	4	4	raid0	raid0	bcache
dmcache-4s0-4h10	4	4	raid0	raid10	dmcache
bcache-4s0-4h10	4	4	raid0	raid10	bcache

Os dados dos experimentos foram coletados através da sumarização apresentada pelo *Fio* e por ferramentas auxiliares, como *iostat* [17] (informações dos dispositivos de I/O do sistema), *SysFS* (dados do BCache) e *dmsetup* [18] (informações do DMCache). Uma rodada inicial de aquecimento de cache de 1.200s foi executada antes de cada caso de teste, sendo descartada dos resultados finais. Cada teste foi executado entre 5 e 15 vezes, até os resultados obtidos apresentarem um coeficiente de variação inferior a 5% ($c_v = s/\bar{x}$).

V. Resultados Experimentais

Nesta Seção apresentamos os resultados experimentais obtidos para os diversos padrões de acesso e configurações

[1]Os testes foram executados apenas para *workloads* aleatórios, uma vez que o caching de acesso sequencial é evitado pelo BCache e DMCache.

escolhidas descritas na seção anterior. Os resultados estão separados por forma de acesso (leitura ou escrita) para facilitar sua análise.

A. Leitura aleatória

A Figura 2 traz os resultados de vazão para leitura aleatória, com todos os tipos de arranjos testados. Barras muito elevadas foram truncadas, com seu valor numérico indicado no topo. O intervalo de confiança (usando a distribuição *t-student*) é indicado no topo de cada barra, na mesma escala. Cada caso de teste mostra três barras, com os os tamanhos das requisições de dados (64KB, 512KB e 4MB).

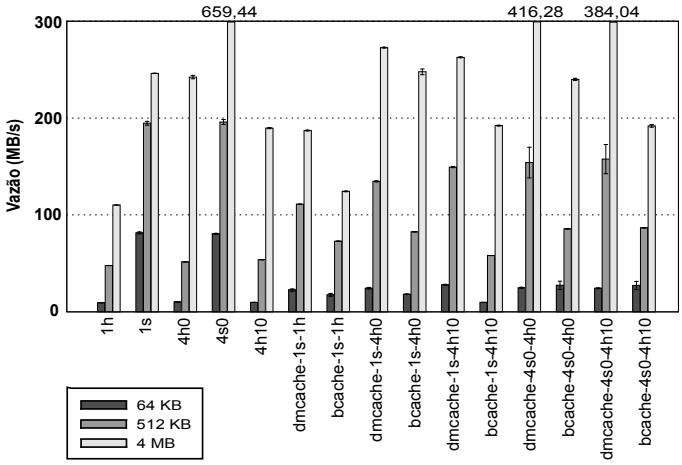

Figura 2. Testes de Leitura Aleatória.

Para leitura aleatória, analisando-se os cinco primeiros casos de testes que envolvem HDDs, SSDs, RAID0 e RAID10, para tamanhos de bloco de 64KB e 512KB, observa-se o mesmo desempenho entre os arranjos *1h*, *4h0* e *4h10*; o mesmo ocorre entre os arranjos *1s* e *4s0*. Isso ocorre porque o tamanho padrão do *chunk* de dados do RAID em software do Linux é de 512KB, ou seja, requisições menores ou iguais a 512KB não são paralelizadas. Como todas as requisições deste caso de testes são aleatórias, o *workload* não se beneficia de *merges* de requisições no escalonador de disco nem de mecanismos de *read-ahead* do sistema de arquivos. O mesmo comportamento se observa nos dispositivos de cache usando o DMCache, para blocos menores ou iguais ao *chunk* do RAID, por exemplo. Numa análise similar, o DMCache apresentou uma variação média na vazão de 7,9% nos casos com blocos de 64KB, 13,5% com blocos de 512KB, mas chegou a 30,9% com blocos de 4MB (nos quais o RAID têm influência).

Apesar disso, diversos casos de leitura aleatória com arranjos de cache obtiveram desempenho superior aos arranjos de HDDs em RAID. Por exemplo, *dmcache-1s-1h* teve desempenho 2,2× superior ao *4h0* e foi 2,3× melhor que *4h10* para 64KB e 512KB. Isso se deve ao melhor desempenho do SSD em relação ao HDD, que contribui para que o arranjos de cache obtenham melhor desempenho que arranjos RAID, especialmente com blocos menores que o *chunk* do RAID. Isso evidencia que usar caches SSD em *workloads* de leitura aleatória melhora o desempenho frente a RAID para casos onde o tamanho médio de requisições seja menor que o *chunk*. Isso porque o paralelismo do RAID não atua nesses casos, mas o ganho do dispositivo SSD tem impacto no desempenho.

Observado esse melhor comportamento dos arranjos de cache, deve-se observar também a vantagem do SSD sobre o HDD. Para os casos de teste, o SSD teve 8,6× melhor desempenho que o HDD para blocos de 64KB, 4,1× para 512KB e 2,2× para 4MB. Em testes separados com blocos de 4KB, o SSD chegou a $\approx 50\times$ melhor desempenho que o HDD para leituras aleatórias; conforme o padrão de acesso torna-se mais sequencial (com blocos maiores, por exemplo), a diferença tende a diminuir.

Pode-se também observar que o DMCache foi superior ao BCache em quase todos os casos de configuração igual ou muito similar. Entre *bcache-1s-1h* e *dmcache-1s-1h*, o DMCache teve um desempenho 28,6% superior para requisições de 64KB, 52,4% para 512KB e 50,4% para requisições de 4MB. Além disso, o DMCache teve um desempenho médio 23,9% acima do BCache em todos os *workloads* com cache. Esse desempenho se deve a uma melhor taxa de acertos do algoritmo de substituição do DMCache, fazendo com que o SSD seja mais utilizado durante os *workloads* e assim melhorando o desempenho global.

B. Escrita Aleatória

A Figura 3 apresenta os resultados de vazão para escrita aleatória. Ao analisar discos isolados (*1h* e *1s*), observa-se que o HDD mantém a mesma vazão da leitura aleatória, devido ao dispositivo ter os mesmos tempo de acesso nessas operações. Já o SSD mostrou diferença entre leituras e escritas, o que remete à assincronia entre essas operações nestes dispositivos.

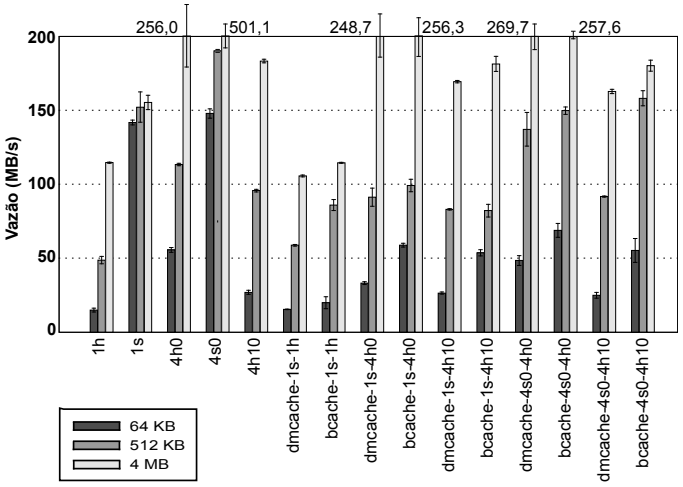

Figura 3. *Escrita Aleatória.*

Além disso, o SSD mostrou poucas diferenças na escrita com blocos maiores, ao contrário do que ocorreu na leitura. Isso ocorre porque todas as operações foram realizadas com a flag *sync* habilitada, o que faz com que uma leitura só possa ocorrer depois que a anterior é completada. Neste caso, tanto HDDs quanto SSDs esperam pelo dado e o tamanho de bloco é determinante. Para a escrita, a maioria dos SSDs trabalha com memórias cache internas, que confirmam a escrita rapidamente e aceleram a vazão, reduzindo o impacto do tamanho de bloco.

Outra diferença entre escritas e leituras aleatórias é a vazão de arranjos RAID para escritas com bloco menor que o *chunk* (64KB e 512KB). Nestes casos, a escrita teve melhor vazão em RAID0 e RAID10, por exemplo, *4h0* foi 3,7× melhor que *1h*

para blocos de 64KB. As solicitações de escritas são entregues ao dispositivo RAID, que as paraleliza quando possível, através de filas e *merges*, gerando ganhos. O mesmo não pode ocorrer na leitura síncrona, mesmo com o uso de RAID, deixando o desempenho refém do *seek time* nos HDDs e da latência normal da operação nos SSDs. Para HDDs em RAID10 (*4h10*), o desempenho seguiu o mesmo padrão apresentado anteriormente, com uma leve redução do desempenho de escrita aleatória em relação ao RAID0, por conta das confirmações de escrita do nível 1 de RAID presente no RAID10. Como esperado, o RAID10 (*4h10*), por exemplo, obteve metade (26 MB/s) do desempenho do RAID0 (*4h0*, 54 MB/s) e foi 2× melhor que um HDD (14,6 MB/s), para blocos de 64KB.

Em relação aos arranjos de cache, nos testes de escrita aleatória ocorre um comportamento inverso à leitura aleatória: o BCache é superior ao DMCache na maioria dos testes. Esse comportamento ocorre porque o DMCache prioriza leituras em relação a escritas, por conta das limitações dos SDDs, como *write-amplification* e limite de escrita por bloco. Além disso, os SSDs possuem desempenho médio de escrita aleatória bem inferior ao desempenho de leitura. Para validação, submetemos a teste requisições de 4KB (tamanho padrão de bloco nos sistemas de arquivos Ext4), o que aumenta o *seek time* e diminui o desempenho dos HDDs, e verificamos que o SSD foi 88× superior ao HDD na leitura e apenas 37× na escrita.

O DMCache regula a prioridade de escritas e leituras com os parâmetros *write_promote_adjustement* (default = 8) e *read_promote_adjustement* (default = 4). Antes de serem trazidos para o dispositivo de cache, os blocos precisam ter ao menos 8 acessos em escrita ou 4 em leitura. Os efeitos desses parâmetros são percebidos nos resultados apresentados. Apesar do resultado inferior do DMCache na escrita, em situações reais a junção da escrita e leitura aliada à importância dos dados mantidos em cache determina um melhor desempenho para o DMCache. Os demais casos de testes entre arranjos de cache seguem os mesmos padrões de desempenho obtidos para a leitura aleatória, seguindo o desempenho de *dmcache-1s-1h* e *bcache-1s-1h* com as diferenças de desempenho dos arranjos RAID envolvidos, já discutidas anteriormente.

VI. Trabalhos Relacionados

Em [7], é apresentado um algoritmo de cache específico para segundo nível, independente do uso de SSDs; os resultados apresentados no trabalho utilizam *traces* proprietários e tamanhos de cache muito reduzidos. Em [9], os autores se limitam a testes em cache de primeiro nível; os resultados são apresentados somente com referências a *I/O hits* para *workloads* sintéticos. Em [19], os autores apresentam uma técnica de cache secundário com SSDs entre disco e memória principal, que monitora padrões de acesso a disco e identifica blocos mais acessadas de acordo com a região do disco em que se encontram. Por fim, [20] estuda o impacto dos tamanhos de caches, gerência de metadados e fluxos de dados sobre diversas configurações hierárquicas de armazenamento, incluindo memória RAM, SSDs e HDDs. Estes dois últimos trabalhos são de aplicação específica a sistemas de bancos de dados.

VII. Conclusão

Este trabalho compara duas implementações recentes de cache em SSD para Linux: BCache e DMCache. Além disso, analisa o impacto desses dispositivos diante de abordagens tradicionais como RAID. Os testes revelaram um grande impacto do tamanho das requisições frente ao tamanho de *chunk* do RAID e como os SSDs amenizam essa situação para casos de escrita aleatória. Para os arranjos de cache de segundo nível, observam-se vantagens do DMCache sobre o BCache na leitura aleatória e resultado inverso na escrita aleatória, devido a parametrizações *default* do DMCache, por considerar as limitações de escrita dos SSDs. Mesmo assim, o DMCache tem um desempenho global superior ao BCache, por conta da leitura. Em alguns casos, ambas as implementações de cache mostraram-se como uma melhor alternativa em relação ao RAID de HDDs, como na leitura aleatória de blocos menores que o tamanho de *chunk*.

Este trabalho está sendo expandido com mais casos de testes para os arranjos estudados, usando *workloads* mais próximos aos utilizados em ambientes reais, através de experimentos que simulam servidores de e-mail, de arquivos e bancos de dados. Outras possíveis extensões seriam testes com *traces* extraídos de sistemas reais e a análise do comportamento dos caches para diferentes tamanhos.

Referências

[1] J. Thornber, "DM-Cache," https://www.kernel.org/doc/Documentation/device-mapper/cache.txt, 2014.

[2] K. Overstreet, "Linux BCache," http://evilpiepirate.org/git/linux-bcache.git/tree/Documentation/bcache.txt, 2003.

[3] D. A. Patterson, G. A. Gibson, and R. H. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)," in *SIGMOD Conference*, 1988.

[4] M. Cornwell, "Anatomy of a solid-state drive," *Communications of the ACM*, vol. 55, no. 12, pp. 59–63, Dec. 2012.

[5] J. Tjioe, A. Blanco, T. Xie, and Y. Ouyang, "Making garbage collection wear conscious for flash SSD," in *IEEE Intl Conference on Networking, Architecture and Storage*. IEEE, 2012, pp. 114–123.

[6] I. Koltsidas and S. D. Viglas, "Data management over flash memory," in *ACM SIGMOD Intl Conference on Management of Data*. ACM, 2011, pp. 1209–1212.

[7] Y. Zhou, Z. Chen, and K. Li, "Second-level buffer cache management," *IEEE Trans. on Parallel and Distributed Systems*, vol. 15, no. 7, 2004.

[8] S. Jiang and X. Zhang, "LIRS: an efficient low inter-reference recency set replacement policy to improve buffer cache performance," *Sigmetrics Performance Evaluation Review*, vol. 30, no. 1, pp. 31–42, 2002.

[9] N. Megiddo and D. S. Modha, "ARC: A self-tuning, low overhead replacement cache," in *USENIX Conference on File and Storage Technologies*, 2003, pp. 115–130.

[10] P. Rocha and L. Santos, "Uma análise experimental de desempenho dos protocolos de armazenamento AoE e iSCSI," in *SEMISH/SBC*, 2012.

[11] P. Koutoupis, "Advanced hard drive caching techniques," *Linux Journal*, no. 233, Sep. 2013.

[12] "Device-mapper Resource Page," https://www.sourceware.org/dm/, 2014.

[13] "Ultrastar 7k3000 datasheet," HGST - Western Digital, Tech. Rep. DSUS7K3011EN-03, 2012.

[14] "Intel SSD 320 Series (160GB, 2.5in SATA 3Gb/s, 25nm, MLC) Datasheet," Intel Corporation, Tech. Rep., 2015.

[15] "FIO," http://freecode.com/projects/fio, 2014.

[16] Jens Axboe, "Measuring IOPS," http://www.spinics.net/lists/fio/msg00830.html, 2015.

[17] "iostat," http://linux.die.net/man/1/iostat, 2015.

[18] "dmsetup," http://linux.die.net/man/8/dmsetup, 2015.

[19] M. Canim, G. A. Mihaila, B. Bhattacharjee, K. A. Ross, and C. A. Lang, "SSD bufferpool extensions for database systems," *Proc. VLDB Endowment*, vol. 3, no. 1-2, pp. 1435–1446, Sep. 2010.

[20] I. Koltsidas, S. Viglas, and P. Buneman, *Flashing Up the Storage Hierarchy*. University of Edinburgh, 2010.