ETERNAL: Uma estratégia eficiente de tolerância a falhas utilizando memória não volátil

Davi B. Gomes[1], **Angelo Brayner**[1], **Javam C. Machado**[1]

[1]Universidade Federal do Ceará
CEP 60440-900 – Fortaleza – CE – Brazil

davi.braga@lsbd.ufc.br, brayner@dc.ufc.br, javam.machado@lsbd.ufc.br

Abstract. *In-memory DBMSs have proven to be an efficient alternative for managing large volumes of data. They're characterized by using RAM as their primary storage medium. However, such systems need nonvolatile storage to ensure the durability of its transactions. Byte addressable non-volatile memories (NVRAM) are ideal candidates for securing such a property because they have access times close to traditional RAM memory but still guarantee data persistence. In order to address the problem of transaction durability in main memory DBMSs, this paper proposes ETERNAL, a durability architecture that makes efficient use of nonvolatile memory, improving the process performance of persistence in such DBMSs. Experiments reveal that ETERNAL provides a higher throughput than the write-ahead logging (WAL) approach. It is also noteworthy that, even using nonvolatile memory as a storage medium, the WAL approach does not have a mechanism to deal with the scenario in which nonvolatile memory runs out.*

Resumo. *SGBDs em memória principal têm se mostrado como alternativa eficiente para o gerenciamento de grandes volumes de dados. Eles caracterizam-se por utilizar memória RAM como seu meio de armazenamento primário. No entanto, tais sistemas precisam de armazenamento não-volátil para garantir a durabilidade de suas transações. Memórias não-voláteis endereçáveis por byte (NVRAM) são candidatas ideais para assegurar tal propriedade, pois possuem tempo de acesso próximo ao de memória RAM tradicional, mas ainda assim garantem a persistência dos dados. A fim de atacar o problema da durabilidade de transações em SGBDs em memória, este trabalho propõe ETERNAL, uma arquitetura de durabilidade que faz o uso eficiente de memória não-volátil, melhorando o desempenho do processo de persistência em tais SGBDs. Os experimentos revelam que ETERNAL provê um throughput superior a abordagem de escrita antecipada em log (WAL). Destaca-se ainda o fato, que, mesmo utilizando memória não-volátil como meio de armazenamento, a abordagem WAL não possui um mecanismo para lidar com o cenário em que a memória não volátil se esgota.*

1. Introdução

O uso de sistemas de bancos de dados em memória principal (SBDMP) tem crescido significativamente. Esse fato ocorre principalmente em cenários cujas aplicações requerem alto desempenho, no que concerne a altas taxas de *throughput* e a baixo tempo de resposta

de consultas. SBDMPs caracterizam-se pelo fato que seus dados tornam-se persistentes, quando o gerenciador de recuperação descarrega para memória não-volátil uma imagem do banco de dados em memória ou registros de log. Assim, ocorrendo uma queda do sistema, todos os dados, que estavam em memória volátil, podem ser recuperados.

Para garantir a durabilidade de transações confirmadas, a técnica mais utilizada em sistemas de banco de dados convencionais é a gravação antecipada de log (*write-ahead logging* – WAL) na qual qualquer transação só poderá ser confirmada após todas as suas alterações terem sido gravadas antecipadamente em log. Tal técnica, no entanto, pode introduzir uma significante sobrecarga no desempenho de sistemas de banco de dados em memória [Faerber et al. 2017], pois, apesar das transações serem executadas rapidamente em memória principal, precisam necessariamente passar pela etapa de confirmação e escrita de registros no arquivo de log residente em memória não-volátil. Como se sabe, os dispositivos atuais de memória não-volátil, como discos rígidos (HDs), apresentam altos tempos de latência de acesso. Desta forma, para minimizar custo dessa etapa, um possível candidato para o armazenamento persistente do log é a nova classe de memórias não-voláteis endereçáveis por byte.

As memórias não-voláteis endereçáveis por byte (NVRAM) representam uma alternativa para incrementar significativamente o desempenho de sistemas que demandam persistência. Dentre as suas principais características, destacam-se a baixa latência de acesso, bem próxima de uma memória RAM convencional, e o baixo consumo energético, não sendo necessária uma corrente elétrica permanente para garantir a persistência dos dados [Arulraj and Pavlo 2017]. Um dos principais impedimentos ao seu uso generalizado, no entanto, é seu maior custo quando comparado aos discos tradicionais, o que exige um uso eficiente.

Cargas de trabalho OLTP possuem um padrão de acesso *skewed* com relação à idade e ao uso do dado. Tipicamente, apenas alguns registros "quentes"são escritos mais frequentemente, porém, com o passar do tempo, ocorre diminuição da frequência de escrita e tais registros tornam-se "frios", passando a ter menor probabilidade de serem escritos no futuro [Eldawy et al. 2014], eventualmente recebendo alguma leitura. Em termos de desempenho, é interessante que tais registros quentes permaneçam em meios de armazenamento de acesso mais rápidos. Algumas abordagens que procuram diminuir o custo do meio armazenamento levando em conta o *skew* de cargas OLTP em SGBDs de memória principal já foram propostas anteriormente [DeBrabant et al. 2013] [Amora et al. 2018]. Nenhuma, no entanto, tentou aproveitar essa característica no mecanismo de recuperação do sistema de banco de dados, o componente onde encontra-se o gargalo relacionado ao custo adicional da técnica WAL, para armazenamento de registros de log.

Este trabalho propõe ETERNAL, um mecanismo de recuperação que explora as propriedades de armazenamento híbrido para garantir durabilidade de transações confirmadas. Para tanto, o ETERNAL garante que as transações são primeiramente confirmadas e persistidas em um armazenamento intermediário persistente de NVRAM. ETERNAL usa a baixa latência e endereçabilidade por byte da NVRAM para atualizar individualmente a última imagem escrita de um dado. Assim, há uma redução no número de registros de log para várias atualizações sobre um mesmo dado. Adicionalmente, ETERNAL permite o despejo assíncrono do conteúdo da NVRAM para HDD, fazendo com que

a etapa de confirmação das transações não fique diretamente dependente da latência de acesso ao disco.

Este artigo está assim estruturado. Inicialmente, os trabalhos relacionados são apresentados e discutidos na sessão 2. Na sessão 3 são descritos a arquitetura de ETERNAL e os protocolos de confirmação, de despejo em memória não-volátil e de recuperação. Na sessão 4 são discutidos os experimentos e na sessão 5 tem-se a conclusão e propostas de trabalhos futuros.

2. Trabalhos Relacionados

A abordagem dominante de recuperação de falhas utilizada por sistemas baseados em disco é o protocolo ARIES [Mohan et al. 1992]. Em geral, a propriedade de durabilidade e a recuperação em sistemas de memória principal são baseadas em WAL. No entanto, ao examinar os detalhes, a diferença entre os sistemas baseados em disco e os em memória principal é grande. Muitos sistemas em memória principal evitam usar protocolos no estilo ARIES por razões de desempenho [Faerber et al. 2017].

2.1. Recuperação em Banco de Dados em Memória Principal

O subsistema de *logging* de SGBDs em memória principal é otimizado para alto *throughput* e para baixa latência. Como o *I/O* do log é o maior gargalo, esses sistemas procuram reduzir o volume de logs o máximo possível, principalmente em sistemas baseados em disco. Hekaton, por exemplo, realiza o log do tipo *redo only* gravando a versão mais recente do registro apenas para transações garantidas [Diaconu et al. 2013]. Essa abordagem evita escrever informações de *undo* completamente. H-Store/VoltDB usam uma variante enxuta de log lógico chamada *command logging* [Malviya et al. 2014] que registra somente a solicitação de invocação de transação. O registro de log desse esquema consiste unicamente no nome do procedimento armazenado, parâmetros de entrada e o ID da transação. Para minimizar ainda mais a escrita de dados em logs, vários sistemas também evitam os metadados de índices [Faerber et al. 2017]. Apenas as atualizações para o banco de dados são registradas.

2.2. Memória Não Volátil e Recuperação

[Kimura 2015] e [Arulraj et al. 2016] realizaram trabalhos que fizeram uso das capacidades únicas da memória não volátil para recuperação de falhas, mas assumem que a maior parte do conteúdo persistente do banco caberá em NVRAM, não lidando com cenários de armazenamento híbrido. [Huang et al. 2014] propôs NV-Logging, uma estratégia de log em memória não volátil para SGBDs tradicionais em disco, porém, a liberação do armazenamento NVRAM depente de *checkpoints*. *Checkpoints* em SGBDs tradicionais apenas precisam persistir o estado corrente do *buffer* em memória, pois é assumido que grande parte do banco já se encontra em disco. Já em banco de dados em memória principal, todo o banco está em memória, tornando o processo inviável nesse cenário.

3. ETERNAL

A arquitetura do ETERNAL pode ser vista na Figura 1. O banco de dados se localiza em espaço de armazenamento volátil, materializado pela memória principal, enquanto que o espaço de persistência é composto por NVRAM e HDD. ETERNAL divide a NVRAM

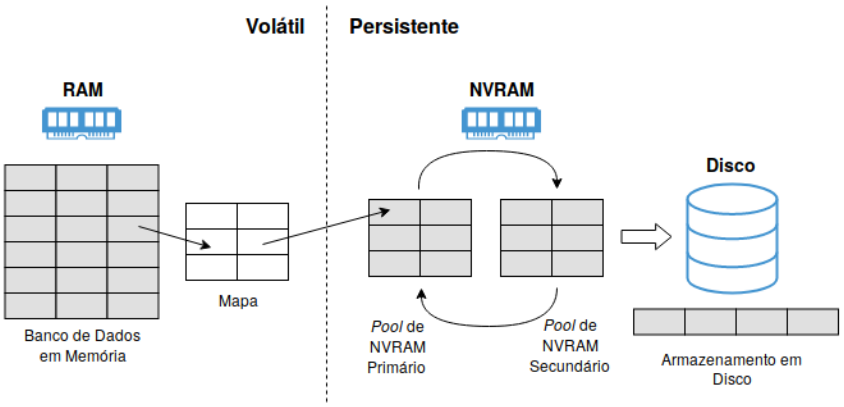

Figura 1. Arquitetura ETERNAL

em *pool* primário e secundário. Há também um mapa ligando registros em memória volátil a endereços de memória não volátil. O mapa indexa as imagens presentes no pool, evitando a necessidade de percorrer a lista encadeada presente em NVRAM no momento da confirmação. Além disso, a latência de acesso da RAM é menor que a da NVRAM, logo, indexar as imagens utilizando um mapa em RAM é mais eficiente. Transações podem realizar dois tipos de operações: leituras e escritas. Inserções, atualizações e deleções são operações de escrita. Para garantir a durabilidade de uma transação, é necessário que as escritas realizadas por ela estejam persistidas em armazenamento durável antes da sinalização de sua confirmação. Em ETERNAL, essa garantia é dada por meio de um mecanismo de lista de imagens.

3.1. Protocolo de Confirmação

Para cada transação em execução, o SGBD manterá uma lista contendo a última imagem de cada registro por ela escrito. Se um registro é escrito mais de uma vez por uma mesma transação, então apenas a sua última imagem desse registro será mantida na lista de imagens. Todo o mecanismo de execução de transações e de preenchimento da lista de imagens é realizado apenas em memória principal. Somente quando uma transação atingir a sua etapa de confirmação, o SGBD garantirá a persistência de sua lista de imagens de maneira atômica. Essa lista será inicialmente persistida no *pool* de NVRAM primário, representado na Figura 1. A transação só poderá ser confirmada após a persistência de sua lista de imagens.

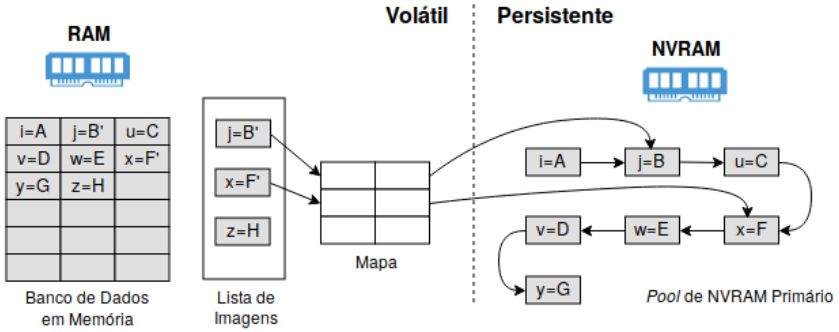

Figura 2. Etapa de confirmação

O processo detalhado de confirmação é ilustrado na Figura 2. Ao entrar na etapa de confirmação de uma transação, o SGBD verificará, para cada registro com imagem presente na lista de imagens, se há ou não uma imagem referente a esse registro presente no *pool* primário de NVRAM. Essa verificação é feita pelo mapa, uma estrutura de dados em memória volátil que mapeia o identificador único do registro à localização de sua imagem no *pool* primário. Caso não seja encontrada uma entrada de imagem do registro, essa imagem será adicionada ao *pool* primário de NVRAM e o mapa em memória volátil será atualizado com essa nova entrada. Caso seja encontrada uma entrada, então a localização da imagem no *pool* de NVRAM é recuperada por meio do mapa e o seu valor será atualizado no local. A etapa de confirmação ocorre de maneira serial, ou seja, apenas uma transação poderá realizá-la por vez.

No exemplo da Figura 2, temos uma transação que atualizou o valor do registro j de B para B', o valor do registro x de F para F' e inseriu uma novo registro z com valor H. A lista de imagens dessa transação, portanto, possui uma imagem para cada registro escrito. Os registros j e x possuem entradas no mapa, indicando que há imagens desses registros no *pool*, já z não possui entrada, indicando que será necessário adicionar uma nova imagem. Observa-se que o *pool* de NVRAM é organizado no formato de uma lista encadeada. A única situação na qual ela precisará ser percorrida é no momento da recuperação após falha. Essa estrutura de dados foi escolhida para que a inserção de novas entradas seja realizada com o menor custo possível, sendo o mapa em memória volátil utilizado para indexar a localização das imagens no *pool*. É possível perceber que o banco de dados em memória principal já possui os dados atualizados com as escritas da transação, mas o *pool* de NVRAM primário ainda possui a imagem dos valores antigos de B e F, além de não possuir uma imagem de H. A etapa de confirmação do exemplo representada na Figura 2, portanto, consiste em atualizar, de forma atômica, as imagens dos registros j e x presentes no *pool* de NVRAM primário, bem como adicionar a imagem de H. O estado do *pool* de NVRAM primário após o término da etapa de confirmação é ilustrado pela Figura 3.

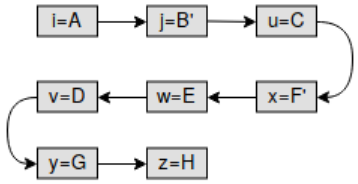

Figura 3. *Pool* de NVRAM após etapa de confirmação

É importante notar que ETERNAL usa NVRAM como meio de persistência dos dados necessários à etapa de confirmação. Isso é uma vantagem, pois significa que o tempo de resposta das transações estará sujeito à latência de acesso da NVRAM, que é próxima à latência de acesso da memória RAM tradicional.

3.2. Uso Eficiente de NVRAM

Um aspecto em que ETERNAL é significativamente melhor que as abordagens baseadas em WAL é com relação ao uso eficiente das memórias do tipo NVRAM, pois nossa abordagem tira proveito da capacidade de endereçamento por byte desse tipo de dispositivo. Em uma típica abordagem baseada em log, novas entradas são concatenadas a calda do

log conforme registros vão sendo escritos por transações. A Figura 4 mostra a evolução de um log no caso em que um registro é inserido inicialmente com valor A, em seguida tem o seu valor atualizado para B e finalmente é atualizado para C.

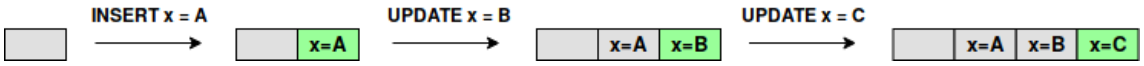

Figura 4. Abordagem baseada em WAL

No exemplo da Figura 4, um registro passa por três operações de escrita, gerando três entradas em log. O caso onde um mesmo registro é escrito repetidas vezes por diversas transações não é incomum, de fato esta é uma característica típica de cargas de trabalho OLTP. Nesse tipo de carga, registros acessados mais recentemente têm maiores chances de serem acessados em um futuro próximo, tais registros também são chamados de "quentes". Eventualmente, com o tempo, a taxa de acesso desses registros cai e eles passam a ser chamados de "frios". [Eldawy et al. 2014]

Já no exemplo da Figura 5, é demonstrado como se dá o uso do armazenamento na abordagem ETERNAL. Como as imagens de registros escritos são armazenadas em NVRAM, um meio endereçável por byte, então é possível indexá-las a nível de registro. Quando um mesmo registro é escrito mais de uma vez, basta atualizar a sua última imagem presente no *pool* primário de NVRAM, pois apenas a última imagem é necessária ao protocolo de recuperação.

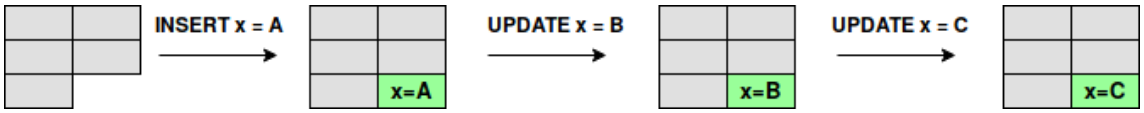

Figura 5. Abordagem ETERNAL

O principal aspecto no qual ETERNAL supera as abordagens baseadas em WAL está relacionado ao comportamento do SGBD no cenário em que o armazenamento NVRAM se esgota. Como demonstrado acima, ETERNAL não precisa manter várias imagens de um mesmo registro que é repetidamente escrito. Essa característica garante que, para uma mesma carga de trabalho OLTP, a situação de esgotamento de NVRAM é menos frequente em ETERNAL que nas abordagens baseadas em WAL, diminuindo drasticamente a necessidade de recorrer a armazenamentos mais lentos. Além disso, ETERNAL implementa um mecanismo assíncrono de despejo de dados de NVRAM para disco.

O uso de NVRAM é imprescindível em ETERNAL, pois, diferente da NVRAM, HDDs e SSDs são endereçáveis por bloco, portanto, os dados devem ser agrupados em páginas, ocasionando overhead adicional em memória principal. Atualizações in place em dispositivos de bloco podem facilmente alcançar diversas páginas, incorrendo no custo do *flush* de várias páginas em um dispositivo que já possui alta latência. Pior ainda, o padrão de acesso a essas páginas não é necessariamente sequencial, penalizando ainda mais o tempo de acesso em tais dispositivos.

3.3. Despejo em Disco

Conforme ilustrado na Figura 1, a arquitetura ETERNAL é composta por dois *pools* de NVRAM. Como já discutido anteriormente, o *pool* de NVRAM primário é utilizado na

persistência da lista de imagens durante a etapa de confirmação. O pool secundário, no entanto, é utilizado como um armazenamento temporário para o despejo assíncrono de dados de imagem para disco. Quando o armazenamento primário é completamente preenchido por imagens de registros, o SGBD realiza uma rotação de *pools*. O *pool* secundário assume o papel de *pool* primário, recebendo as novas cópias de imagens. Já o *pool* primário, que estava completamente preenchido com imagens, assume o lugar do secundário, sendo varrido por uma *thread* auxiliar que despeja dados de imagens em disco. Caso o *pool* primário se esgote enquanto o *pool* secundário ainda estiver realizando um despejo, então as transações serão bloqueadas até o início da próxima rotação.

A *thread* de despejo varre a lista encadeada de imagens presente no *pool* de NVRAM. Cada imagem de registro é copiada para um arquivo sequencial em disco. Antes de iniciar a cópia das imagens, um marcador é inserido no arquivo para indicar o início de um novo despejo, outro marcador será inserido ao término do despejo. O SGDB garante que o *pool* secundário será limpo para uma nova rotação apenas após o término da cópia de todas as imagens de registro para disco, assegurando que todas as imagens presentes no *pool* estarão persistidas em disco antes da limpeza. Na Figura 6, temos uma ilustração de um despejo no qual as imagens dos registros u, v, x e y foram copiadas para disco. É possível ver também os marcadores de início do despejo (ID) e de fim do despejo (FD).

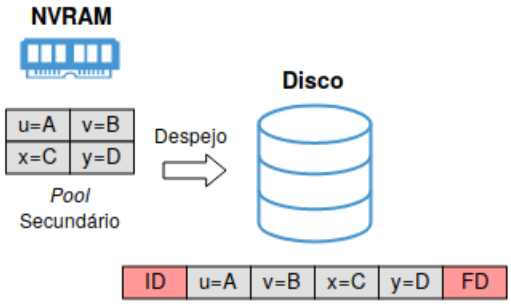

Figura 6. Despejo em disco

A grande vantagem do mecanismo de *pools* rotativos é que a etapa de confirmação das transações não fica diretamente dependente da latência de acesso ao disco, pois a etapa de confirmação é realizada diretamente em NVRAM, tirando proveito da sua menor latência e diminuindo o tempo de resposta das transações.

3.4. Protocolo de Recuperação

Quando ocorre uma queda do SGBD, todo o conteúdo presente em memória não volátil é perdido. No caso específico de SGBDs em memória principal, isso significa que todo o banco deverá ser carregado em memória novamente. Em ETERNAL, o protocolo de recuperação funciona carregando a última imagem transacionalmente consistente de cada registro do banco, não sendo necessário um histórico de atualizações. Como o protocolo de confirmação garante que toda transação deve persistir as imagens de todos registros por ela escritos em armazenamento durável, então existem três localizações possíveis para as imagens: o *pool* de NVRAM primário, o *pool* de NVRAM secundário e o arquivo sequencial em disco. O processo de recuperação consiste em percorrer todos essas localizações, identificar a imagem mais atual de cada registro e carregá-la no banco.

O protocolo de recuperação deve levar em conta que é possível existir mais de uma versão de imagem de um mesmo registro nos diferentes meios de armazenamento. O mapa utilizado no protocolo de confirmação assegura que existe apenas uma imagem de um mesmo registro no *pool* de NVRAM primário. O mesmo vale para o *pool* de NVRAM secundário, mas não para o arquivo sequencial em disco. Mais de uma imagem de um mesmo registro em disco pode ocorrer quando um registro cuja imagem já havia sido despejada é escrito novamente, gerando uma nova imagem no *pool* de NVRAM primário que eventualmente será despejada novamente para disco.

A versão mais nova da imagem de um registro estará sempre no *pool* de NVRAM primário, pois é nele onde as imagens são inicialmente persistidas durante a etapa de confirmação. Em seguida, temos o *pool* secundário, com versões das imagens presentes no momento do despejo. Por último, temos as imagens presentes no arquivo sequencial, já despejadas em disco, com imagens mais antigas no início do arquivo e mais novas no fim. Para garantir que a versão final dos registros após a recuperação será a mais nova, o algoritmo de recuperação inicia o carregamento do banco a partir das imagens mais antigas, as sobrescrevendo sempre que encontrar uma imagem mais nova de um registro já carregado.

Os marcadores de início e de fim de despejo são utilizados para garantir a atomicidade de toda a operação quando uma falha ocorre em meio a um despejo. Se, no momento da recuperação, for detectado um despejo com marcador de início, mas sem um marcador de fim, então significa que esse despejo não foi concluído completamente, possivelmente devido a uma falha. Nesse caso, todas as imagens relativas a esse despejo serão ignoradas. Como o pool secundário só é limpo após a inserção do marcador de de fim de despejo em disco, então temos a garantia de que essas imagens ainda estarão persistidas no pool secundário, assegurando a correta recuperação.

4. Experimentação

Apresenta-se agora a análise experimental comparando o desempenho de ETERNAL com a abordagem estado da arte baseada em WAL e utilizada em [Malviya et al. 2014] para banco de dados em memória principal. Para tornar a comparação mais justa, a abordagem concorrente será adaptada para utilizar NVRAM como meio de armazenamento para seu log. Tal como ocorre em ETERNAL, o log da abordagem WAL será despejado para disco caso a NVRAM seja completamente preenchida. Chamaremos essa abordagem de NVWAL. Os experimentos foram realizados em uma máquina Intel Core i7-7800X com 6 núcleos e 12 threads rodando a uma frequência base de 3.5GHz e 128GB de memória SDRAM DDR4, com sistema operacional Ubuntu 18.04. A NVRAM utilizada no experimento é emulada [Maciejewski 2017].

Assim como acontece em trabalhos que se diferenciam fundamentalmente da maneira como SGBDs tradicionalmente funcionam, ETERNAL e NVWAL foram implementados como *engines standalone* em C++ [Kimura 2015]. Para fins de avaliação, ambos os *engines* de recuperação foram conectados a um banco de dados em memória simples baseado na coleção std::unordered_map da biblioteca padrão de C++. Tanto o banco quanto as *engines* foram compilados em um mesmo binário junto com uma implementação do *benchmark* YSCB em C++. O código fonte está disponível em [Gomes 2019].

4.1. Carga de Trabalho

O *benchmark* usado no experimento é o YCSB [Cooper et al. 2010], utilizado para modelar aplicações OLTP. Como o objetivo do experimento é avaliar o desempenho do sistema de banco em cenários de escrita, não são utilizadas operações de leitura, além disso, cargas de leitura não afetam o subsistema de recuperação. Dois tipos de cargas foram submetidas: carga de atualização, com 100% das operações sendo de atualizações e carga de inserção, com 50% das operações sendo de inserções e 50% sendo de atualizações. O YCSB também permite selecionar a distribuição estatística que modela a frequência de acesso dos registros do banco. As distribuições mais utilizadas para modelar cenários do mundo real são a zipfian e a latest [Cooper et al. 2010]. Em todos os experimentos o banco é inicializado com 12GB de registros. A quantidade de NVRAM emulada é de 2GB, equivalente a aproximadamente 17% do tamanho inicial do banco. Em ETERNAL, cada *pool* de NVRAM tem 1GB. É importante mencionar que a atomicidade das operações realizadas em NVRAM são garantidas pela biblioteca libpmemobj por meio de logs de *undo* e que seu código fonte é aberto [Balcer 2017].

4.2. Análise dos Resultados Experimentais

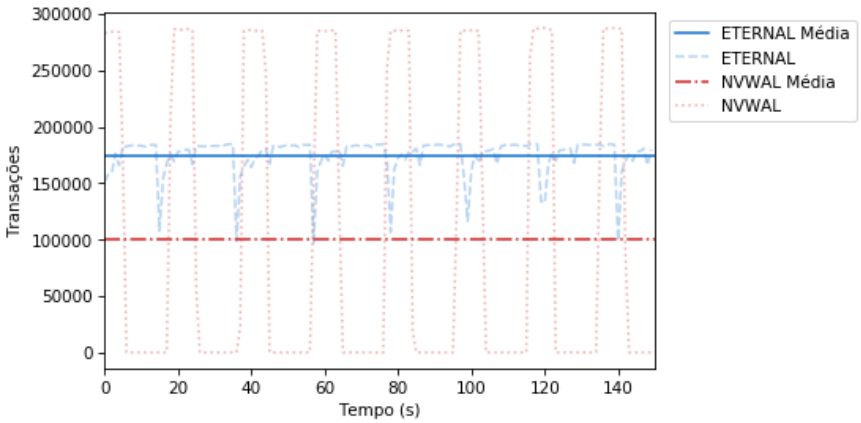

Figura 7. *Throughput*: atualização com distribuição zipfian

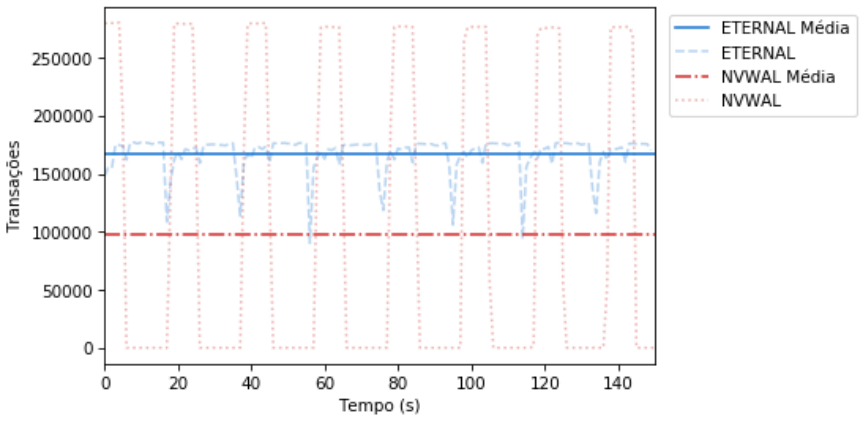

Figura 8. *Throughput*: atualização com distribuição latest

Nas Figuras 7 e 8 são apresentados os gráficos de *throughput* que mostram o número de transações realizadas a cada segundo na carga de atualização utilizando as distribuições zipfian e latest respectivamente. Temos também duas linhas representando o *throughput* médio total do experimento para as duas abordagens. ETERNAL apresentou um *throughput* médio de 174327 TXs/s contra 99792 TXs/s de NVWAL com a distribuição zipfian. Com a distribuição lastest, ETERNAL apresentou um *throughput* médio de 167650 TXs/s contra 97939 TXs/s de NVWAL . O *throughput* total dos experimentos de ETERNAL foi 73% maior quando comparada ao *throughput* de NVWAL.

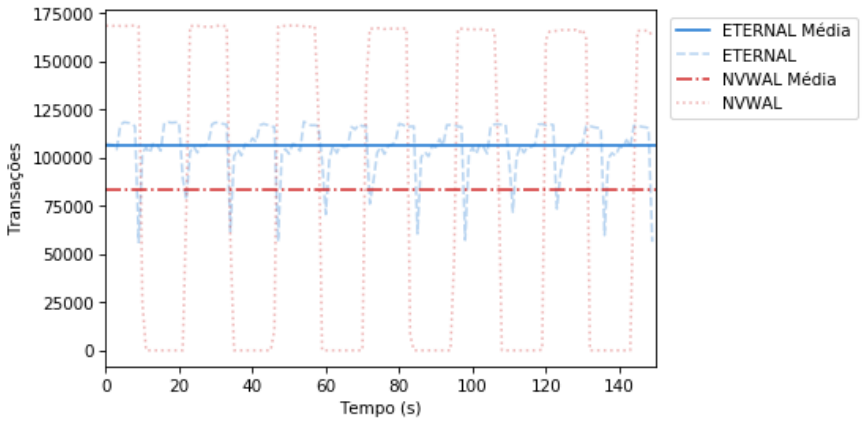

Figura 9. *Throughput*: inserção com distribuição zipfian

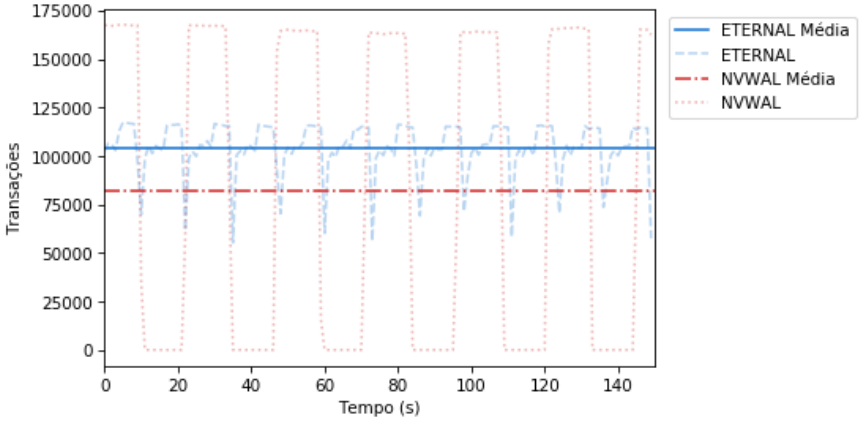

Figura 10. *Throughput*: inserção com distribuição latest

Nas Figuras 9 e 10 são apresentados os gráficos de *throughput* que mostram o número de transações realizadas a cada segundo na carga de inserção utilizando as distribuições zipfian e latest respectivamente. ETERNAL apresentou um *throughput* médio de 106000 TXs/s contra 83579 TXs/s de NVWAL com a distribuição zipfian. Com a distribuição lastest, ETERNAL apresentou um *throughput* médio de 104495 TXs/s contra 82438 TXs/s de NVWAL . A *throughput* total dos experimentos de ETERNAL foi 26% maior quando comparada ao *throughput* de NVWAL.

Analisando os gráficos, é possível perceber que, na média, ETERNAL atinge *throughputs* significativamente maiores que NVWAL. É possível perceber também que

NVWAL apresenta, em alguns momentos, valores de *throughput* maiores que os de ETERNAL. Essa perda momentânea de desempenho pode ser explicada em parte pela sobrecarga adicional causada pela uso do mapa para acessar as imagens no *pool* primário de NVRAM. Outro fator a ser considerado é a sobrecarga gerada pelos logs de *undo* da biblioteca libpmemobj, necessários para garantir a atomicidade das operações realizadas em NVRAM. Ambas distribuições apresentaram resultados semelhantes porque ambas possuem *skew*. O comportamento seria diferente em uma distribuição uniforme. Uma carga uniforme, no entanto, não modela bem padrões de acesso OLTP. O desempenho em cargas com inserções é menor, pois cada inserção representa necessariamente um dado novo e portanto não repetido, aumentando a quantidade de despejos.

A vantagem de ETERNAL se dá no momento em que a NVRAM se esgota, pois, ao fazer o uso eficiente da tecnologia, a abordagem consegue diminuir a quantidade de dados despejados para disco. Por causa desse mecanismo, o *pool* se esgota com menor frequência, permitindo um despejo sem contenção realizado por uma *thread* de fundo. NVWAL despeja, de maneira síncrona, uma quantidade muito maior de dados, o que explica as quedas de *throughput* apresentadas nos gráficos da abordagem. Caso NVWAL utilizasse *pools* de NVRAM, de nada adiantaria, pois ambos os pools seriam preenchidos rapidamente, ocasionando bloqueios de espera pelo despejo. Uma forma de demonstrar a quantidade de acesso a disco evitada por ETERNAL é comparar o tamanho do arquivo sequencial despejado em disco com o tamanho do log gerado por NVWAL. Ao final do experimento, com a carga de atualização, o arquivo sequencial de ETERNAL possuía um tamanho de 6GB, já o log gerado por NVWAL possuía 18GB. Portanto, NVWAL realizou 3 vezes mais acessos a disco que ETERNAL, o que também explica o seu desempenho geral inferior.

5. Conclusão e Trabalhos Futuros

Este trabalho apresentou ETERNAL, uma arquitetura de recuperação que faz o uso eficiente das memórias não voláteis endereçáveis por byte. ETERNAL projeta e implementa um protocolo de confirmação, um protocolo de recuperação e um mecanismo assíncrono de despejo de dados de NVRAM para disco. Quando comparado a outras abordagens baseadas em WAL, ETERNAL demonstra ganhos significativos de *throughput* de até 73% em cargas de trabalho OLTP, como pode ser constatado nos resultados experimentais.

A partir de ETERNAL, muitos trabalhos futuros podem ser realizados. O *pool* secundário é esvaziado durante a realização de um despejo. É possível melhorar esse mecanismo permitindo que imagens de registros mais acessados permaneçam no *pool*, evitando o custo de adicionar novas imagens do mesmo registro nas próximas rotações. Novos experimentos podem ainda ser realizados em outros *benchmarks* como o TPC-C.

Agradecimentos

Esta pesquisa foi parcialmente financiada pela Lenovo, como parte do seu investimento em pesquisa e desenvolvimento de acordo com a Lei de Informática, pela CAPES (Número do Processo: 1697962) e pelo LSBD/UFC.

Referências

Amora, P. R. P., Teixeira, E. M., Praciano, F. D. B. S., and Machado, J. C. (2018). Smartltm: Smart larger-than-memory storage for hybrid database systems. In *XXXIII Sim-*

pósio Brasileiro de Banco de Dados, SBBD 2018, Rio de Janeiro, RJ, Brazil, August 25-26, 2018., pages 13–24.

Arulraj, J. and Pavlo, A. (2017). How to build a non-volatile memory database management system. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1753–1758. ACM.

Arulraj, J., Perron, M., and Pavlo, A. (2016). Write-behind logging. *Proceedings of the VLDB Endowment*, 10(4):337–348.

Balcer, P. (2017). http://pmem.io/2015/06/15/transactions.html. Acessado em: 27/05/2019.

Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R. (2010). Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154. ACM.

DeBrabant, J., Pavlo, A., Tu, S., Stonebraker, M., and Zdonik, S. (2013). Anti-caching: A new approach to database management system architecture. *Proceedings of the VLDB Endowment*, 6(14):1942–1953.

Diaconu, C., Freedman, C., Ismert, E., Larson, P.-A., Mittal, P., Stonecipher, R., Verma, N., and Zwilling, M. (2013). Hekaton: Sql server's memory-optimized oltp engine. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 1243–1254. ACM.

Eldawy, A., Levandoski, J., and Larson, P.-Å. (2014). Trekking through siberia: Managing cold data in a memory-optimized database. *Proceedings of the VLDB Endowment*, 7(11):931–942.

Faerber, F., Kemper, A., Larson, P.-Å., Levandoski, J., Neumann, T., Pavlo, A., et al. (2017). Main memory database systems. *Foundations and Trends® in Databases*, 8(1-2):1–130.

Gomes, D. B. (2019). https://github.com/davibrg/nvrec_ycsb. Acessado em: 26/08/2019.

Huang, J., Schwan, K., and Qureshi, M. K. (2014). Nvram-aware logging in transaction systems. *Proceedings of the VLDB Endowment*, 8(4):389–400.

Kimura, H. (2015). Foedus: Oltp engine for a thousand cores and nvram. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 691–706. ACM.

Maciejewski, M. (2017). https://pmem.io/2016/02/22/pm-emulation.html. Acessado em: 27/05/2019.

Malviya, N., Weisberg, A., Madden, S., and Stonebraker, M. (2014). Rethinking main memory oltp recovery. In *2014 IEEE 30th International Conference on Data Engineering*, pages 604–615. IEEE.

Mohan, C., Haderle, D., Lindsay, B., Pirahesh, H., and Schwarz, P. (1992). Aries: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. *ACM Transactions on Database Systems (TODS)*, 17(1):94–162.