C-018

書換え耐性が低い NAND flash メモリ向けのキャッシュアルゴリズムの提案

A Proposal of Cache Algorithm for Low Endurance NAND Flash Memories

松田 慎平†

Shinpei Matsuda

1. はじめに

人類が生み出すデータ量は加速度的な増大を続けておりこれらのデータを収集・保存・加工するデータセンターの消費する電力の増大を如何に抑制するかは、気候変動の進行を食い止める意味でも重大な課題となっている。データ爆発は動画共有・配信サービスや SNS といった近年発達した Web サービスによって誘発されているが、データセンターに蓄積されているであろうそれらの膨大なコンテンツは、孤独と退屈から逃れたいという人類の悪足掻きがビット列にエンコードされているかのようである。

NAND Flash メモリは高密度に集積可能で、HDD と比較して消費電力が少ないことから、ストレージメモリデバイスとして今後も中心的な役割を果たすことが期待されている。NAND Flash メモリの高集積化は、微細化や３次元化といったプロセス技術の開発と、一つのメモリセルに 1bit 以上の情報を記憶させる多値記憶技術によって推し進められてきた[1]。NAND Flash メモリは絶縁膜のトンネル電流によって書き込みと消去を行っており、書き込みや消去動作が行われるたびに、絶縁膜の劣化が引き起こされるため、そのメモリセルには書き換え可能な回数に上限が存在する。この問題は、メモリセル全体を均等に使用する技術であるウェアレベリングや、DRAM あるいは書き換え耐性の高い 1bit/cell(SLC)のメモリセルをキャッシュとして用いることで対処されてきた。しかし、多値記憶技術が 5bit/cell や 6bit/cell 技術の登場が見込まれる段階に至り、書き換え可能回数の上限は 100 回程度、もしくはそれ以下となると予想され、ウェアレベリングや単純なキャッシュだけでは対応しれなくなることが懸念される。多値記憶技術の進展が停滞すれば、NAND Flash メモリの高密度化・大容量化を継続することがより困難となり、データセンターの消費電力を抑制することは難しくなるだろう。

本報告では、多値記憶技術の進展に伴いさらに深刻化する NAND Flash メモリセルの書き換え耐性の低下に対処するため、複数の異なる記憶密度の NAND Flash メモリセルを組み合わせたハイブリッドストレージを如何に構成すべきかの戦略を議論する。まず、公開されているストレージへのメモリアクセスのトレースを解析し、その統計的な性質について議論する。次に、２階層のストレージよりも３階層のストレージのほうが、書き換え耐性が極めて低いメモリに適していることを主張する。次に、３種類のメモリ間で、データをどのように保存するべきかを考察し、選択的 Eviction というコンセプトを提案する。最後に、選択的 Eviction を実現するためのキャシュ置換アルゴリズムの一例として、LRU-k キャッシュ置換アルゴリズムを応用することを提案し、実際のトレースを用いて評価を行う。

†所属なし　No affiliation

2. メモリへのアクセスが有する統計的性質

まず、ストレージへのメモリアクセスが有する一般的な性質について議論するために、Microsoft Research Cambridge が公開しているトレースファイルを解析した[2]。これらのトレースファイルは、Microsoft の Windows サーバにおける、１週間分の HDD へのアクセス履歴のログであり、リクエストが発行されたタイムスタンプ、リクエストの始点となるブロックアドレス、リクエストの長さ、リクエストの種類（Write or Read）といった情報がまとめられている。

2.1 ストレージトレースの可視化

CPU からストレージへの参照は 512Byte のセクタ単位で発行されるが、この報告では 32 セクタ(16kByte)単位に換算して解析を行った。この単位をページと呼ぶことにする。図１にストレージへのアクセスを横軸にタイムスタンプ、縦軸にページのアドレスとして散布図で可視化した。視覚化することで、メモリアクセスのいくつか性質を明確に見て取ることができる。例えば、prxy_0 トレースには、ページアドレス 18000 番付近に、右肩上がりの斜めの線が繰り返し並んでいる様子が確認できるが、これはある範囲のアドレスがループでアクセスされている様子を表している（図１）。

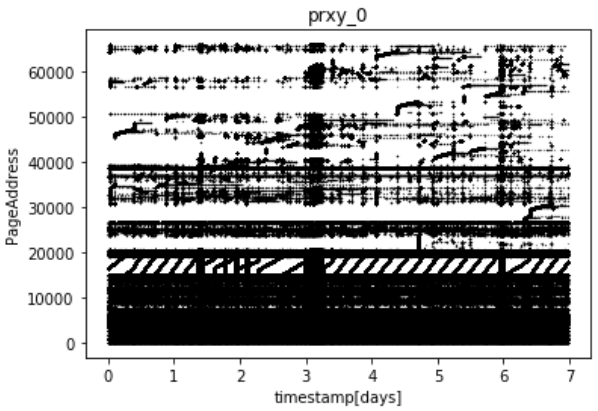

図１　ストレージへのメモリアクセスの可視化(prxy_0)

また、web_0 トレースには、縦に長い線のようなパターンがときどき発生しているのが見えるが、これは連続したデータをスキャンするようなアクセスが発生したことを意味している（図２）。

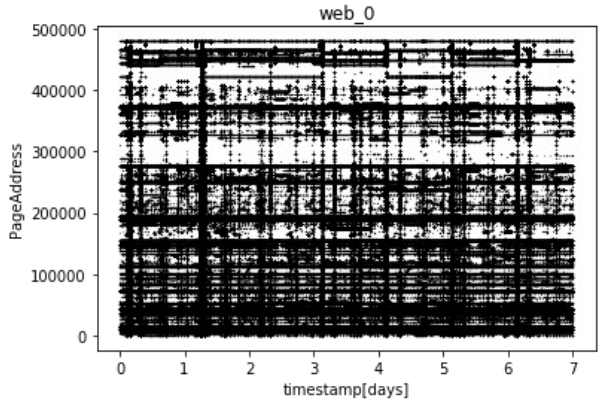

図２　ストレージへのメモリアクセスの可視化(web_0)

2.2 Zipfian 分布

次に、web_0 トレースに対してページ毎にアクセス回数をカウントして、横軸をアクセスが多い順に並び替えたときの順位(Rank)、横軸をアクセス回数（Frequency)としてプロットした。図 3 を見ると、Rank に対して Frequency が反比例する、すなわち Rank の-1 乗に比例して Frequency が小さくなる傾向があることが見て取れる。このような統計分布は、Zipf の法則や Zipfian 分布として知られており、人間が使う自然言語の単語の出現回数や都市の人口など、社会現象によく見られる統計的な分布である[3]。Zipfian 分布への一致が悪いトレースも存在するが、一般にはストレージへのメモリアクセスは、Zipfian 分布に従うと期待しても良い。

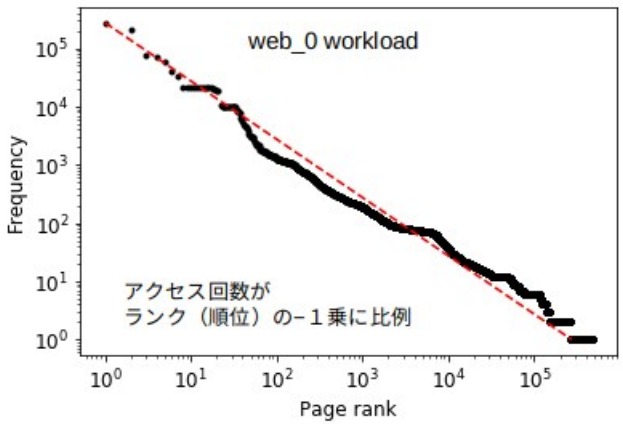

図３　ストレージへのメモリアクセスに現れる Zipfian 分布

後の議論のために、アクセスの頻度を示す用語を慣例に従って定義する。上で述べたとおり、ストレージ内のページごとにアクセス頻度は大きく異なるが、アクセス頻度が高いことをホット、低いことをコールドなどと、温度に例えて表現することが多いので、本報告もその流儀に従う。

さらに、週に１回しかアクセスされないページをフローズンと呼ぶことにする。web_0 トレースの場合、全アドレスのうち 46%ものページがフローズンである。

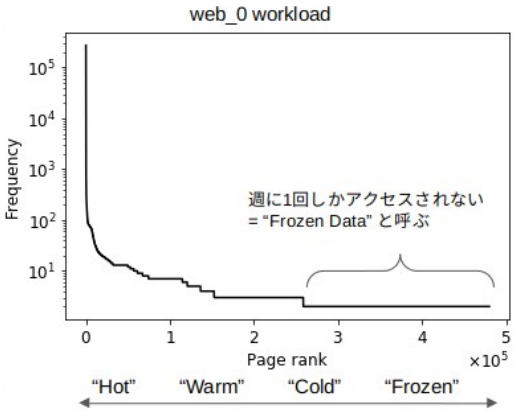

図４　web_0 のアクセス頻度分布とフローズンデータ
（図３のグラフを片対数プロットしたもの）

2.3 時間的局所性

次にあるアドレスがアクセスされてから次にアクセスされるまでに要する時間について調べよう。これを参照間時間(Inter-reference time)と呼ぶことにする。web_0 トレースの参照間時間を集計し、参照間時間と参照回数の関係をプロットした（参照間時間は１秒ごとに区切り、その出現回数をカウントした）。図５から見て取れる通り、同じアドレスが再び参照される確率は、前のアクセスから時間が経つほど急速に低下する傾向が見て取れる。これは時間的局所性と呼ばれるメモリアクセスの重要な性質であり、ストレージへのメモリアクセスでも時間的局所性は成立していることを端的に示している。

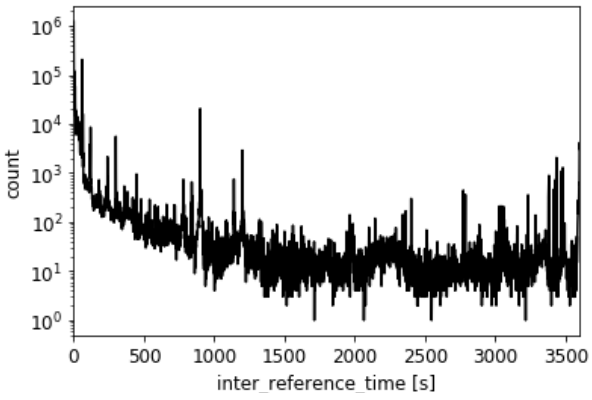

図５　web_0 トレースにおける参照間時間の分布
（前の参照からの経過時間が短いほど
そのアドレスが再参照される確率が高い）

2.4 なぜ Zipfian Distribution を取るのか？

本論から外れるが、なぜストレージメモリへのアクセスの　パターンが自然言語と同じ統計的性質を示す傾向にあるのかを考察したい。なお、特に裏付けがある考察ではな

いことは予め断っておく。

Zipfian 分布は、より一般には冪乗則（Power law)として知られており、スケールフリーネットワークの頂点の次数や、都市の人口と大きさの順位、山崩れや火災、地震といった災害の規模と頻度など、自然科学と社会科学の両方で広範に観察される統計分布である。ストレージメモリへのアクセスに一番性質が近い事象は、おそらく自然言語であろう。CPU とストレージの間でやり取りされる Write/Read リクエストは、人間同士の会話に類似している。人間は周囲の事物に名前を与えることで、世界を切り分ている。自然言語とは、周囲の人間と共有された事物を表す音声的・視覚的な記号とその記号の操作体系である。共有されていることで、周囲の人間との音声による会話や、文字を使った視覚的な方法での意思疎通が可能となっている。自然言語における単語の出現回数の分布は、突き詰めると人間の脳の特性がもたらすものだろう。脳の神経網が複雑ネットワーク性を有しているのであれば、個々のニューロンが発火する確率は、周囲の神経細胞との接続数で決まると考えられる。図５で示したような時間的局所性は、神経細胞の発火確率が時間とともに指数関数的に減少することと対応しているのだろう。複雑ネットワークの特徴である頂点の次数の統計的分布が冪乗則に従うという性質が、自然言語の単語の出現回数として現れたものと考えられる。すなわち、自然言語は人間の神経網の構造が外部化したものと言えるのではないか？[4] 一方で、コンピューター・プログラムは人間が定めた「決まりごと」としてのシステムが記述されているに過ぎず、自然言語と同じく、脳の神経網の構造が何らかの形で反映されても不思議はない。結局、コンピュータのメモリアクセスパターンを見ることは、非常に遠回りな視点から脳の神経網の構造を観察していることになるのではないだろうか？以上が筆者の個人的な考察である。

近年、東京大学の田中久美子から、自然言語の統計的な普遍性に関する詳細な研究が上梓されている[5]。同著作では、複雑系科学からの視点を伺うことができる。

3. キャッシュアルゴリズム

フォン・ノイマン型コンピュータが発明されて以来、メモリデバイスには一つのトレードオフが存在した。すなわち、「高速なメモリは容量が小さく高価で、安価で大容量なメモリは遅い」ということである[6]。70 年あまりが経過した現在でも、このトレードオフは、コンピュータシステムにおいて深刻なボトルネックであり続けている。このボトルネックに対処するために、メモリ階層及びキャッシュというアイデアが考案され、コンピュータの様々な階層で使われてきた[7]。例えば、CPU は DRAM メインメモリへのアクセス遅延を緩和するために SRAM キャシュを内部に備えているし、オペレーティング・システム（OS)には、ストレージメモリへの遅延を隠蔽するために DRAM によるページキャッシュが用意されている。

キャッシュは下位のメモリに対して高速であるが、その容量は下位のメモリに対して小さいので、書き込まれたデータでキャッシュが満杯になった際にキャッシュの中にあるいずれかのデータを追い出して空き容量を作る必要がある。この際キャッシュに保存されたデータの中から、「不要な」データを追い出す（Evict）必要があるが、このキャッシュから追い出されるページ（犠牲ページ：Victim Page）を選び出すアルゴリズムをキャッシュ置換アルゴリズムとか、単にキャッシュアルゴリズムと呼ぶ。将来アクセスされる可能性が高いページをキャッシュに残し、可能性が低いページを追い出すようにすれば、低速な下位のメモリを参照する回数が低減できるので、キャッシュを搭載することにより全体的にメモリへのアクセスを高速化することができる。キャッシュアルゴリズムの役割を端的に言えば、キャッシュに残すべきホットなページと、残す価値の低いコールドなページを見分けることである。

3.1 OPT アルゴリズムと LRU アルゴリズム

キャッシュに保存されたデータに対して参照が発生することをキャッシュヒットと呼ぶが、キャッシュの性能は全アクセスに対してキャッシュヒットが発生した割合（キャッシュヒット率）で決まる[8]。キャッシュヒット率は、そのキャッシュシステムが処理するワークロードだけでなく、キャッシュ置換アルゴリズムにも依存しているが、その上限は Bélády の OPT アルゴリズム（MIN アルゴリズム、千里眼アルゴリズムとも呼ばれる）で与えられる[9]。OPT アルゴリズムは、キャッシュが満杯となった際、次のアクセスが最も遠い未来となるページを犠牲ページとして選ぶアルゴリズムである。

OPT アルゴリズムでは将来に発生するメモリへのアクセスを予め知っている必要があるため、多くの場合は実装が不可能であるが、コンピュータにおいて最もよく採用されるキャッシュアルゴリズムである Least Recently Used (LRU)アルゴリズムでは、犠牲ページとして最後のアクセスが最も遠い過去であったページが選ばれる。OPT アルゴリズムと LRU アルゴリズムは、「未来」と「過去」を入れ替えた、いわば鏡像関係にあることに注意したい。LRU アルゴリズムは、将来参照されるページを時間的局所性と呼ばれるメモリアクセスの性質に基づいて予想している。時間的局所性という言葉の定義を教科書から引用すると、「ある項目が参照された場合、その項目がまもなく再び参照される確率が高い」というメモリアクセスが有する、普遍性が高い性質のことである[7]。LRU アルゴリズムは、実装コストとキャッシュ性能のバランスに優れており、第一選択肢となるキャッシュアルゴリズムである。DRAM すらまだ登場していなかった時代に、キャッシュアルゴリズムに関する最も重要な仕事が完成していたという歴史的な事実は、控えめに言っても驚嘆に値する。

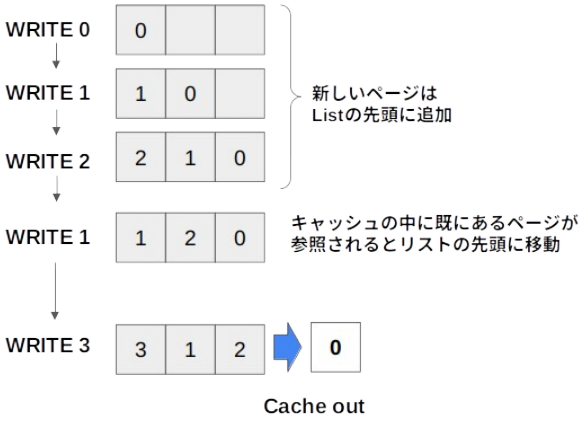

図 6　LRU アルゴリズムの挙動

3.2 スキャン耐性を有するキャッシュアルゴリズム

時間的局所性の原則に基づきつつ、LRU アルゴリズムを発展させて乗り越えようとする試みも数多く報告されている。LRU アルゴリズムの弱点として、コールド・スキャンと呼ばれるアクセスのパターンに弱いことが挙げられる LRU アルゴリズムではスキャン的なアクセスがあったとして、コールドなページもホットなページも区別せずにキューの先頭から詰めていくが、連続したコールドなページに対して参照が発生した場合、キャッシュの中に保存すべきホットなページが追い出されてしまい、キャッシュミスが多発するようになる。（上で紹介したとおり、web_0 トレースに見られる、縦に長い棒のようなアクセスパターンが典型である。）時間的局所性の原則は活かしつつ、このようなスキャン動作が発生した際には、コールドなページをホットなページに優先して追い出されるような動作が望ましい。このようなキャッシュアルゴリズムの性質を「スキャン耐性がある」という。

1993 年、O. E. Neil らは最後のアクセスではなく最後から k 番目（多くの場合は k= 2）の過去のアクセスのタイムスタンプが最も古いページを追い出すアルゴリズム（LRU-k）を提案した[10]。LRU アルゴリズムでは、コールドなページもホットなページも区別せずにキューの先頭から詰めていくが、web_1 トレースの 2 日目にあったように、連続したアドレスに一括してアクセスが発生した場合、キャッシュの中に保存すべきホットなページが追い出されてしまい、キャッシュミスが多発するようになる。LRU-k アルゴリズムを採用したキャッシュにコールドなページが書き込まれた場合を考えよう(k=2 とする)。コールドなページはアクセス頻度が低いため、最近アクセスされてキャッシュに書き込まれたページは、 1 番目の過去のアクセスのタイムスタンプが新しいが、2 番目前のアクセスのタイムスタンプは古いだろう、逆に、ホットなページではアクセスが頻繁にあることを考えると 2 番目の過去のタイムスタンプも、最近のものであると期待できる。すなわち、最新のアクセスのタイムスタンプではなく、 2 番目の過去のタイムスタンプが古い順に犠牲ページを選び出す LRU-k アルゴリズムでは、時短的局所性を強化することでページがホットかコールドかをある程度の精度で見分けることが可能であり、コールドと思われるページを優先して追い出すことができるのである。

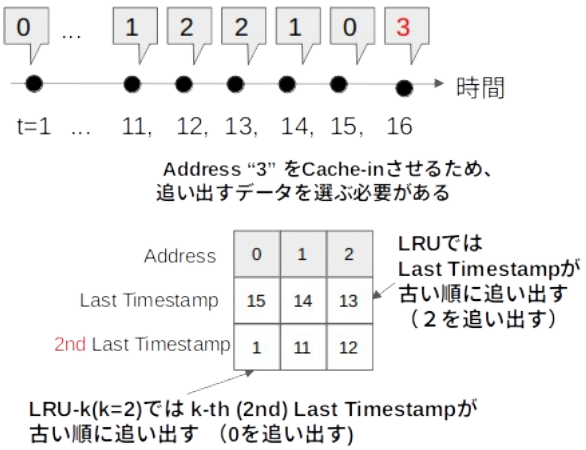

図 7　LRU-k アルゴリズムの挙動

LRU-k アルゴリズムのメリットは、スキャン耐性を有している点であるが、LRU アルゴリズムでは o(1) で理想的には犠牲ページを選ぶことが可能であるのに対して、LRU-k アルゴリズムは犠牲ページを選び出すのに o(logn) の時間を要することから(LRU-k では犠牲ページを選ぶために二分探索が用いられる)、よりシンプルなアルゴリズムが望まれていた。T. Johnson らは、非常にシンプルなアイデアでこの問題を解決しつつ、スキャン耐性を両立させる提案を行った[11]。２ Q アルゴリズムでは、A1 と Am と呼ばれる 2 つのキューを有しており、新しくキャッシュに書き込まれるページは、 A1 キューに書き込まれる。A1 キューは FIFO（First-In, First-Out）キューであり、もし書き込まれたページがアクセスされることなく A1 キューの末尾に到達したら、ページはキャッシュから追い出される。A1 キューの末尾に至るまでにページが再度参照された場合は、Am キューに書き込まれる。Am キューは LRU であり、Am キューにページが存在する間にページがまた参照された場合、当該ページは Am キューの先頭に再度移動させられる。コールドなページがキャッシュに書き込まれた場合は、Am キューに入ることなく A1 キューを通過してキャッシュから追い出される。ホットなページは A1 キューにある間に参照されて、Am キューに移されているため、コールドなページが大量にキャッシュに書き込まれても、Am キューに存在するホットなページは影響を受けない。また、キャッシュが満杯になった場合は、 2 つのキューの末尾のページを追い出せば良いので、犠牲ページの選択も高速に実行することができる。

LRU-k や 2 Q では、チューニングが必要なパラメータが存在することが問題であった。N. Megiddo らによって 2 つの LRU リストを持ち、それぞれのサイズを自動的に調整する ARC というアルゴリズムが提案されている[12]。

4. ストレージキャッシュ

この節では、ストレージにおけるキャッシュ技術について簡単にまとめる。

4.1 DRAM による揮発性キャッシュ

上で述べたように、ストレージへのアクセス遅延を隠蔽するためにオペレーティングシステムが管理する DRAM キャッシュが用いられており、Linux ではページキャッシュと呼ばれている。あるいは、NAND Flash メモリからなる SSD には、メモリコントローラが管理する DRAM を搭載されており、その一部がキャッシュとして使われている。DRAM をキャッシュとして用いる場合、DRAM が揮発性メモリであるため、書き込まれたデータが意図しない電源 OFF によって失われないよう、DRAM キャッシュに書き込まれたデータを定期的にストレージに書き出す必要がある。

4.2 不揮発性キャッシュ

NAND Flash メモリを HDD の不揮発性キャッシュとして使う技術は、まだ SSD をシステムドライブ（オペレーティングシステム等を保存するためのストレージ）として使用するには高価だった頃に広く使われており、Windows における Ready Boost 技術や、Linux における dm-cache ターゲットなどが例として挙げられる。なお、NAND Flash メモリを HDD に対するキャッシュとして用いる場合、書き換え回数の制限が問題となるため、dm-cache では MQ(Mullti-Queue)と呼ばれるキャッシュアルゴリズムを採用している[13]。

NAND Flash メモリに対するキャッシュ技術として、ストレージクラスメモリ（SCM）を不揮発性キャッシュとして用いた Hybrid SSD[14]として、同グループから数多く報告されている。不揮発性キャッシュの利点は、DRAM キャッシュと違い、データを定期的にストレージに書き出さなくても良いことである。DRAM キャッシュでも、ストレージへの書き込み（読み出し）回数を軽減することはできるが、不揮発性キャッシュを用いる方が、原理的にストレージへの書き込み回数を低減することができる。これは、書き換え回数が有限である NAND Flash メモリにとっては重要なメリットである。

4.3 仮想メモリ

最後に、仮想メモリについて言及する。仮想メモリではコンピュータが搭載しているメモリ領域よりも広いメモリを搭載しているようにアプリケーションに見せかけることができるが、メインメモリが満杯になった際、メインメモリにあるデータの一部をストレージの定められた領域（Unix ではスワップ領域と呼ばれる）に一時的に退避させ、メインメモリに空き容量ができたら再度メインメモリに戻すことを行っている。このとき、メインメモリからストレージへと退避させるページを選ぶ際にキャッシュアルゴリズムが用いられている。過去には、コンピュータにとってメインメモリの容量が足りないことのほうがストレージが遅いことよりも問題であったため、ストレージに関するキャッシュアルゴリズムの初期の応用例は、ストレージを高速化するためのキャッシュではなく、仮想メモリ技術の方が例が多いのだろう。このことを了解しておくと、過去のキャッシュアルゴリズムに関する論文が探すときに有用である。例えば、文献[15]の仮想メモリの章にはキャッシュアルゴリズムの黎明期の論文が数多く紹介されており、キャッシュアルゴリズムの発展をたどる上で参考になる。

5. 書き換え耐性が低い NAND Flash メモリ向けのキャッシュアルゴリズムの戦略

5.1 書き換え耐性が低い NAND Flash メモリを含むメモリ階層の構成案

まず、キャッシュ＋ストレージからなる 2 階層の構成が書き換え耐性の低い NAND Flash メモリに適していないことを主張する。なお、ここでは DRAM キャッシュは考えずに、SLC メモリなどの不揮発性キャッシュのみ考察する。上で述べた通り、SLC キャッシュでは定期的にデータを Evict する必要がないため、下位のメモリへの書き込みリクエストが発行されるのを大幅に抑制できると期待される。実際、3~4bit/cell の程度書き換え耐性のあるストレージに対しては、SLC キャッシュは高速化のみならず、書き換え耐性の長寿命化の観点からも重要な技術であると言える。しかし、5bit/cell や 6bit/cell といった書き換え耐性が著しく低いメモリにおいては、書き換え耐性の低さが、寿命を律速してしまう恐れがある。２階層の構成では書き換え耐性の低さをカバーするには大量のキャッシュを積むか、予備の領域をより多く用意する（オーバプロビジョニング）ことが必要となり、高密度化の恩恵を相殺してしまうことが懸念される。

上で述べた通り、典型的なストレージへのメモリアクセスは Zipfian 分布に従うと期待される。Zipfian 分布に従うということは、アプリケーションが頻繁にアクセスするアドレスはごく一部のアドレスに偏っており、大半のアドレスはほとんどアクセスされることはないことを意味している。web_1 トレースを例に出すと、アドレス空間のうち 47%までが１週間に１回しかアクセスされていない。書き換え回数が著しく低いメモリセルであっても、これらほとんどアクセスされることがないページ（フローズンページ）を保存するためのストレージとしての使用には耐えるだろう。一方、フローズンページよりはアクセス頻度が多いが不揮発性キャッシュに常に留まるほどにホットではないページは、3~4bit/cell などのある程度書き換え耐性があるメモリセルを割り当てるべきである。まとめると、SLC によるキャッシュを備え、3bit/cell や 4bit/cell といったある程度書き換え耐性が備わったメモリセルの他に、フローズンページの保存用の低書き換え耐性のメモリセルを有する構成が適していると考える。以下の議論では、書き換え耐性が高く高速な SLC などのメモリセルを上位メモリ、ある程度の書き換え耐性を備えるメモリセルを中位メモリ、書き換え耐性が著しく低いメモリセルを下位メモリと呼ぶこととする。

5.2 ３階層のストレージにおけるキャシュアルゴリズムに求められること

上・中・下位３階層のストレージにおける制御アルゴリズムについて検討する。単純に考えれば、書き換え耐性の高い上位メモリを Level1(L1)キャッシュ、中位メモリを Level2 (L2) キャッシュとした２段階のキャッシュ設けるのが良さそうだが、この構成だと、中位メモリでのガーベジコレクションが問題となるだろう。NAND Flash メモリは bit 単位の上書きができないため容量が不足するとガーベージコレクションによって新しい空き容量を用意する必要があるが、ガーベージコレクションが頻発すると、NAND Flash メモリの応答性の大幅な低下や、書き換えによるメモセルの悪化を引き起こす。更に悪いことに、２階層のキャッシュを構成した時、中位メモリのガーベージコレクションは上位メモリのそれよりも効率が悪いと考えられる。L1 キャッシュにあるデータは時間的局所性が高い可能性が高く、L1 キャッシュに存在するページに Write リクエストがヒットすることで、当該ページは無効ページとなり、ガーベージコレクションの際に空き容量として解放される。一方で、L2 キャッシュに存在するページは L1 キャッシュに比べて、時間的局所性が低く、上書きされる可能性が低い。結果、L2 キャッシュの容量が足りなくなってガーベージコレクションが走った際、解放されるべき無効ページが少なく、頻繁に容量不足を起こしてはその度にガーベージコレクションが走ることが懸念される。L2 キャッシュにフローズンなページが書き込まれることは、それらが下位メモリに追い出されるまで L2 キャッシュの容量を圧迫し続けるという意味で、最悪のシナリオである。

以上の考察からフローズンページは、中位メモリはスキップして直接下位のメモリセルに書き込まれることが望ましいと言える。すべてのデータは一旦はキャッシュに書き込まれ、上位メモリのキャッシュからフローズンページが追い出される場合は、中位メモリをスキップして下位メモリに書き出すような動作をするキャッシュアルゴリズムが３階層の構成のストレージには適しているだろう。

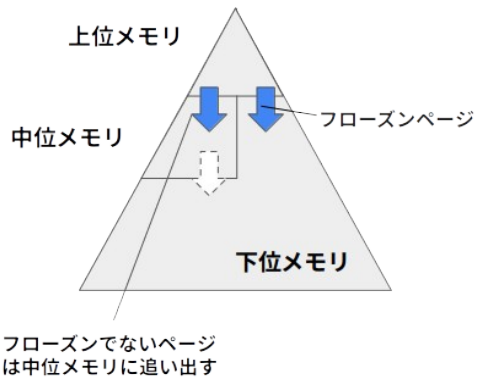

図８　今回提案する NAND Flash メモリによる３階層のストレージと選択的 Eviction の模式図

上位メモリによるキャッシュが満杯になった際に、犠牲ページがフローズンだった場合、中位メモリをスキップして直接下位メモリに追い出される動作を「選択的 Eviction」と呼ぶことにしよう。

6. 選択的 Eviction

この節では、前節で述べた選択的 Eviction を実現するためのキャッシュアルゴリズムを検討する。

まず、LRU アルゴリズムは、選択的 Eviction を実現する上では選択肢とならないことを指摘しておく。何故ならば、LRU キャッシュアルゴリズムには、キャッシュに残すべきページと、残すべきでないページを見分けることはできても、キャッシュから追い出すページがフローズンかどうかを見分ける情報を有していないからである。選択的 Eviction を実現するにはキャッシュからページを追い出す際に、そのページがほとんどアクセスされることがないフローズンなページか、そうでないかを見分けることが必要である。ここで、キャッシュアルゴリズムの節で述べてスキャン耐性に注目する。スキャン耐性があるキャッシュアルゴリズムは、コールドなページを優先し追い出す機構を備えていた。よって、よりコールド（つまりフローズンなページ）と判定されて優先して追い出されるページを下位メモリに、フローズンではないと判定されたページを中位メモリに追い出すようにすれば良さそうである

ここでは、LRU-k アルゴリズムによって選択的 Eviction のモデルを実装した。LRU-k アルゴリズムで管理されるキャッシュにフローズンページが書き込まれることを考える。この場合、フローズンページがキャッシュに書き込まれた時は、当該のフローズンページが初めてアクセスされた時である。（１週間分のストレージへのアクセスのトレースにおいて、１回しかアクセスされないページをフローズンと呼ぶことにしたことを思い出そう）。LRU-k（ここでは k=2 とする）アルゴリズムにおいては、初めてアクセスされるページには、2 番目の過去のアクセスのタイムスタンプ、といった量は定義できない。LRU-k アルゴリズムでは、これらの初めてアクセスされたページは最優先で追い出される対象となる。そして、１週間という期間に限定して考えるならば、キャッシュに書き込まれた全てのフローズンページは、初めてアクセスされたページということになる。よって、LRU-k アルゴリズムにおいて、２番目の過去のタイムスタンプが未定義のページを、中位メモリをスキップして下位メモリに追い出すようにすれば、選択的 Eviction の動作が実現できる。

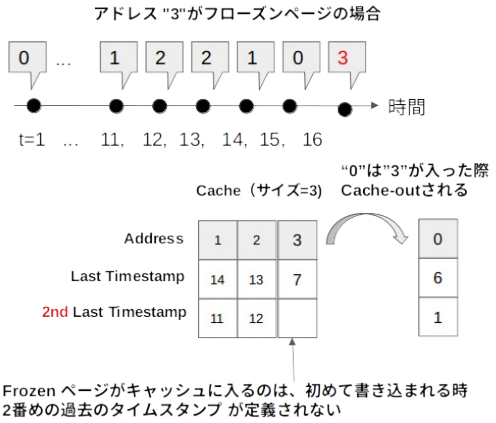

図 9　LRU-k アルゴリズムによるキャッシュに
フローズンページが書き込まれた場合の動作
（アドレス 3 は追い出される時、フローズンと判定）

このとき、フローズンデータでないページも最初は下位ページに追い出されることが副作用として発生するが、2 回目のアクセス以降は、中位メモリに追い出されるようになる。このようにすれば、下位メモリで生じる書き換えを大幅に削減でき、Level 2 キャッシュの構成とした場合に問題になることが懸念された中位メモリにおけるガーベッジコレクションの頻発という問題も回避できるだろう。なお、過去のアクセス履歴をずっと記憶しているならば、下位メモリにおいては、最初の書き込みがなされてから 1 回も書き換えが発生しないことになってしまうが、1 週間よりも古いタイムスタンプのデータを LRU-k か管理するタイムスタンプのテーブルから削除することで、下位メモリへの書き換え回数を確実に週 1 回以下に制限することができる。

LRU-k アルゴリズムを用いて、選択的 Eviction の動作をモデル化した。トレースには web_0 を用いた。なお、選択的 Eviction の動作が確かめたいだけなので、単純にするため、全てのリクエストを Write リクエストとして計算した。キャッスサイズが web_0 データサイズの 1%の場合に、キャッシュである上位メモリ、中位メモリ、下位メモリの書き込み回数をカウントした。全部で 12061632(1.21E+7) ページ分の Write リクエストは一旦 Cache メモリである上位メモリに書き込まれ、そこから中位メモリには 1094702 (1.1E+6) ページ分のデータが追い出され、下位メモリには 478437 (4.78E+55) Page が Evict された。下位メモリに追い出されたページのうち、2.21E+5 ページはフローズンページであり、残りは初めてアクセスされたあと、2 回めのアクセスが発生する前にキャッシュから追い出されたことで、フローズンと判定されたページであった。

以上で示したとおり、LRU-k アルゴリズムの性質を利用することで、選択的 Eviction の動作を実装できることが示された。LRU-k アルゴリズムは、キャッシュ管理のためのテーブルの容量が大きくコストが高いアルゴリズムではあるが、データセンターにおけるストレージサーバの用途であれば実用できるかも知れない。というのも、CPU と接続可能なストレージデバイスの数には上限があるため、ストレージの高密度化のためには、多値記憶技術を極限まで使いこなすことが必要と思われるためである。

7. 結論

この報告では、NAND Flash メモリの多値記憶技術の進展に伴い、書き換え耐性の低下の問題が顕在化するであろうことを指摘した。次に、実際のサーバの HDD へのアクセスのトレースを用いて、ストレージへのメモリアクセスが有する性質について論じた。また、時間的局所性の原則を利用する LRU ベースのキャッシュアルゴリズムについて代表的なものと筆者が考えるものを紹介し、ストレージにおけるキャッシュの先行例について簡単に紹介した。最後に、冒頭で指摘した NAND Flash メモリの書き換え耐性の低下の問題に対処するために、3 階層で構成された NAND Flash メモリが適しているであろうことを主張し、それを使いこなすためのキャッシュアルゴリズム、とりわけ、フローズンページをキャッシュから追い出す際に、中位のメモリをスキップして下位に追い出す、選択的 Eviction というアイデアを提案し、LRU-k アルゴリズムを応用することで Frozen データが Cache に書き込まれた際の挙動で実装可能であることを示した。

ここで示した考察が、加速するデータ爆発に対抗するためのヒントとなるようなことがもしあれば、もちろん望外の喜びではあるが、業界に無関係の人間が余暇で考察したことに過ぎず、そのようなことは文字通り望むべくも無いかも知れない。この文章を通じてキャッシュアルゴリズムやメモリアクセスの統計的性質が示す美しさの一片でも（あるいは、少なくとも筆者がそれらを美しいと感じていることを）伝えることができたならば、目的は果たされたと考えることとしたい。

謝辞

この文章は 2017 年から 2019 年にかけてとある大学の研究室に在籍していた期間に考察した内容を一部含んでいるが、例えばストレージトレースの解析などは、MSRC からダウンロードしたファイルを自分で解析したものであり、如何なるデータ及びコードも、同研究室に由来するものではないことを断っておく。

今更このような発表をしたところで、私には何ら利益はないことは理解しているが、おぼろげに形をなしつつあった「選択的 Eviction」と文中で呼んだアイデアについて、研究室を去った後も、（半ば忌々しくことだったが）無意識に考えてしまっており、3 年の時間を経てようやく文章として吐き出せるに至った、といったところである。3 年の間に、感情と思考が渾然となってしまっており、文章に落とすのに苦労した。これを書き終えた今、もう無意識下で考える必要がなくなったことに心から安堵している。

機械学習ブームに乗ってみたいという、今から思うと大変軽率な理由で応募してしまったが、応募の際に提出した研究計画書と関係があるとは思えないテーマを頂いた時は、この先どうしようかと呆然としたのを覚えている。確かに、現実性に乏しい計画書だったとしか言えないが、ナイーブ

に考えていた私には衝撃が大きかった。方針を考えようと、試しにストレージへのメモリアクセスを可視化した瞬間から、この研究室で行われて来たことには何かしらの数理的な基礎づけが可能だろうという、根拠のない確信を覚えた。体系的な基礎付けができれば、個々の問題を解かずとも「一般解」を与えられるだろうと目論んで解析と学習を続けるうちに、キャッシュというコンピュータ・サイエンスにおける最も本質的なアイデアの美しさと、Zipfian 分布の普遍性と底知れなさに魅力を感じていった。ここで報告した LRU-k を利用した選択的 Eviction というアイデアは、LRU を上回るキャッシュアルゴリズムを考案しようとしているうちに、LRU-k アルゴリズムを再発明してしまったことがきっかけとなったものある。「再発明」の数カ月後、オリジナルの論文を見つけて既知であったことを知り（もちろんオリジナルの方が優れていた）、どうにか活かせないかとぼんやりと考えていたアイデアが元になっている。時間的局所性という普遍的な性質に基づく LRU というアイデアは、おそらくコンピュータサイエンスにおける最も美しいアイデアだと思われるが、時間的局所性を自然に拡張した LRU-k アルゴリズムも私には美しく映ったので、発表せずに忘れ去ることもなかなかできなかった。

研究室を去った後、自力で発表して区切りとしようと考え、2020 年 3 月の電子情報通信学会総合大会に今回の内容の骨子だけを記した要旨をひっそりと投稿した [16]が、Covid-19 の感染拡大によって中止となってしまい発表する機会を失ってしまった。その後、スッキリしないままずるずると放置していたが、2021 年の 11 月に、オンラインの勉強会 KernelVM online4 で本報告の概要を発表する機会を得ることができた。主催者である@nekomatu 氏(Twitter)や、聴講して頂いた参加者の皆様に御礼を申し上げたい。誰とも議論することのなく、自分の胸にしまいこんでいたアイデアについて初めて反応を頂けたことは嬉しかった。文章としてまとめよう、という挫けてしまっていた意志をもう一度与えて頂いたように思う。

最後に、K. Murata に対し心の底からの感謝を表すとともに、彼女の人生に幸福が多く訪れることを祈っている。数年間にわたって、私の心を孤独から救い上げてくれたことを生涯忘れはしないだろう。

参考文献

[1] Hemink, G. J., et al. "Fast and accurate programming method for multi-level NAND EEPROMs." 1995 Symposium on VLSI Technology. Digest of Technical Papers. IEEE, (1995).

[2] MSR Cambridge Traces, http://iotta.snia.org/traces/388

[3] M. Buchanan, “Ubiquity: Why Catastrophes Happen”, (邦訳 ”歴史はべき乗則で動く”) Crown Publishing Group (2002).

[4] 養老 孟司, “唯脳論”, ちくま学芸文庫 (1998).

[5] 田中 久美子, “言語とフラクタル”, 東京大学出版会 (2021).

[6] D. A. Patterson and J. L. Hennessy, “Computer Organization and Design, Fifth Edition: The Hardware/Software Interface”, Morgan Kaufmann Publishers Inc, (2013).

[7] G. Dyson, ”Turing's Cathedral: The Origins of the Digital Universe” (邦訳 ”チューリングの大聖堂”) Pantheon Books, (2012).

[8] Denning,P. J. Denning and K. C. Kahn, “A study of Program Locality and Lifetime Functions,” in Proc. of the fifth ACM Symposium on Operating System Principles, pp. 207-216, Nov. 1975.

[9] L. A. Belady, "A study of replacement algorithms for a virtual-storage computer." IBM Systems journal 5.2 (1966): 78-101.

[10] E. J. O'neil, P. E. O'neil, and G. Weikum, "The LRU-K page replacement algorithm for database disk buffering," in Proc. of ACM SIGMOD Conf., (1993), pp. 297-306.

[11] T. Johnson, and D Shasha. "2Q: a low overhead high performance buffer management replacement algorithm." Proceedings of the 20th International Conference on Very Large Data Bases. (1994).

[12] N. Megiddo, and D. S. Modha, “ARC: A SELF-TUNING, LOW OVERHEAD REPLACEMENT CACHE”. In 2nd USENIX Conference on File and Storage Technologies (FAST) (2003).

[13] “ Guidance for writing policies”, https://www.kernel.org/doc/Documentation/device-mapper/cache-policies.txt

[14] H. Fujii, et al. “x11 performance increase, x6.9 endurance enhancement, 93% energy reduction of 3D TSV-integrated hybrid ReRAM/MLC NAND SSDs by data fragmentation suppression,” in IEEE Symposium on VLSI Circuits Dig. Tech. Papers, Jun. (2012) , pp. 134-135.

[15] 前川 守, “オペレーティングシステム”, 岩波書店, (1988).

[16] 松田 慎平. "超低 Endurance メモリに対応した Cache アルゴリズムの提案." IEICE Conferences Archives. The Institute of Electronics, Information and Communication Engineers, (2020).