【VLDB2013 勉強会】

R1: Emerging Hardware

担当：山室健

Improving Flash Write Performance by Using Update Frequency

Radu Stoica (EPFL), and Anastasia Ailamaki (EPFL)

Improving Flash Write Performance by Using Update Frequency

- A overview
  - SSDの’out-of-place update’を効率的に実現するためwrite IOのworkloadによる影響のモデル化を行い、そのモデルを基にデータ配置のpolicyを提案、結果としてDBのベンチマーク（TPC-C[24]/TATP[19]）において20-75%のGCのoverhead削減を実現

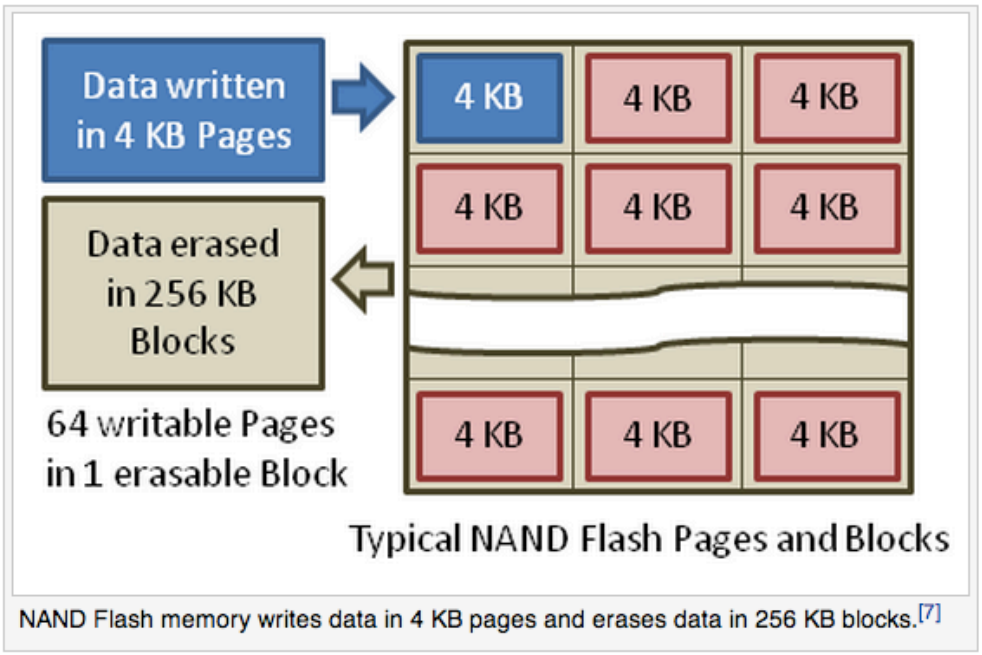
- Contribution
  - 1. 予備ブロック（over-provisioning）/update頻度/データ配置policy/GC戦略の影響のモデル化とシュミレーション
  - 2. workloadに応じたGCのoverheadの定量化
  - 3. k-modal update頻度を用いた効率的なデータ配置policyの提案、parameter-freeで動的なworkloadに対応

Improving Flash Write Performance by Using Update Frequency

- SSDにおけるupdate（pageとblock）

Basic SSD operation [edit]

See also: Flash memory and Solid-state drive



NAND Flash memory writes data in 4 KB pages and erases data in 256 KB blocks.[7]

Due to the nature of flash memory's ope
as it can in a hard disk drive. When dat
start in an erased state so data can be
4–8 kilobytes (KB) in size). The SSD c
flash memory and interfaces with the h
mapping system known as logical bloc
flash translation layer (FTL).[8] When n
already written, the SSD controller will
update the logical mapping to point to t
old location is no longer valid, and will
be written again.[1][9]

Flash memory can only be programme
This is often referred to as the maximu
(P/E cycles) it can sustain over the life
(SLC) flash, designed for higher perfor
typically operate between 50,000 and 1
(MLC) flash is designed for lower cost a

引用元: http://en.wikipedia.org/w/index.php?title=Write_amplification

Improving Flash Write Performance by Using Update Frequency

- uni-modal～k-modal update頻度分析
  - GCのoverheadは「blockをeraseする際の有効pageの割合」
  - SSDの書き込み増分WA（Write Amplification）は

$$WA = 1 + \underbrace{\frac{p_{gc}}{1 - p_{gc}}}_{\text{Cleaning overhead (GC)}}$$

※一般SSD製品のWAは1～10程度
参考: http://www.slideshare.net/InsightTechnology/d17dbhw

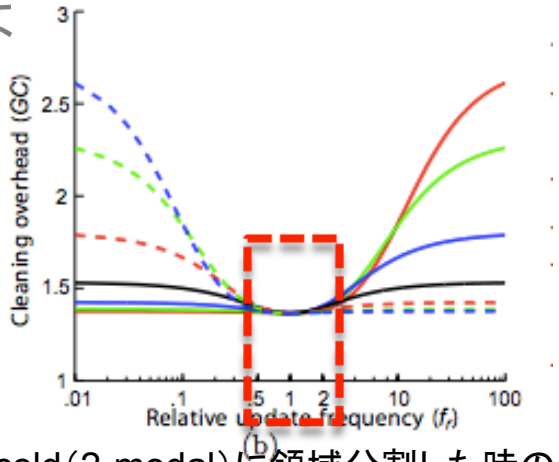


Fig.2から引用: hot/cold（2-modal）に領域分割した時の領域間の相対update頻度とGCのoverheadの関係

- 分析のPractical Implications
  - 2つの領域（2-modal）間でupdate頻度に～x2程度の偏りがあってもGCのoverheadは変化しない

→ update頻度f_r推定の誤差はある程度許容

→ f_r=2, 4,..., 2^kとして分割領域を決定

- 余談）SSD上でのwrite IO操作の注意
  - 非効率なwriteパタンによってもWA値は悪化するケースが存在

例）LMS木実装leveldbのcompaction処理

・https://leveldb.googlecode.com

– O’Neil, P. et al.: The log-structured merge-tree, Acta Informatica 33, 1996

・階層Lのnode（2MiB）から階層L+1上のoverlapするnodes（最大12nodesで24MiB）をmergeする処理でfan-outが多い場合にIOが4倍に悪化

  - mergeのスケジューリングを改善したhyperleveldb

https://github.com/rescrv/HyperLevelDB

Improving Flash Write Performance by Using Update Frequency

- データ配置policyは分析結果を基に構築
  - 1. update頻度推定は同pageへの書き込み間隔を利用したLRUベースの MultiLogと、正確な頻度情報を仮定したMultiLog-Oracle
  - 2. 分析結果から得たGCのoverheadコスト関数(式(11))からupdateの際に promotion判定、GCの際にdemotion判定を実施
  - 3. GCのpolicyは最後に書き込みが行われたblockを採用するLRUベース
    - block内の有効page数が少ないものを選択するGreedyな手法に比べて性能の劣化は少なく、メタ情報を管理する余剰領域もいらないため実現が容易

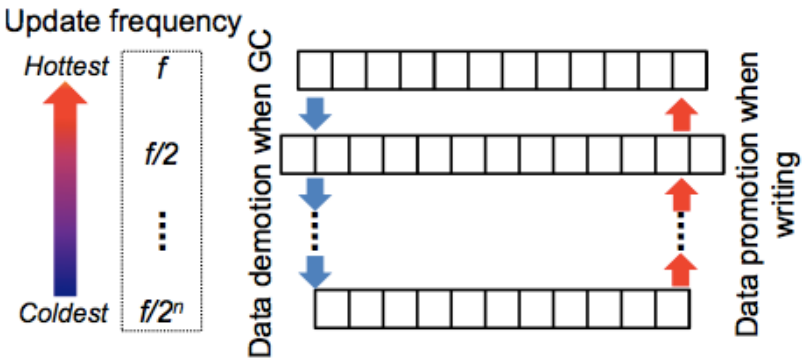


Figure 4: MultiLog data placement

Algorithm 1 Analytical Estimation of Write Overhead

1: Group pages in bins with ±50% update frequency
2: Set number of logs := number of bins
3: Set $\beta_1, \cdots, \beta_k := s_1, \cdots, s_k$
4: *// Iterative space allocation*
5: **repeat**
6: **for** i=1:k **do**
7: re-distribute space between log i and $(i+1)$ *mod* k
8: **end for**
9: **until** no β_i change
10: Calculate GC_{tot}

論文内Fig.4から引用

Improving Flash Write Performance by Using Update Frequency

- 実験概要
  - 実験はTPC-C[24]とNokia TATP[19]のDBベンチマークを利用
    →書き込みが多いworkloadを想定
  - page sizeは4kB/erase blockは64pagesの100GB SSDを想定
  - 予備ブロック割合αとGC overheadの関係を評価
  - 比較手法はNaiveなLRU/Greedyと、既存手法[14][26][12]の3パタン

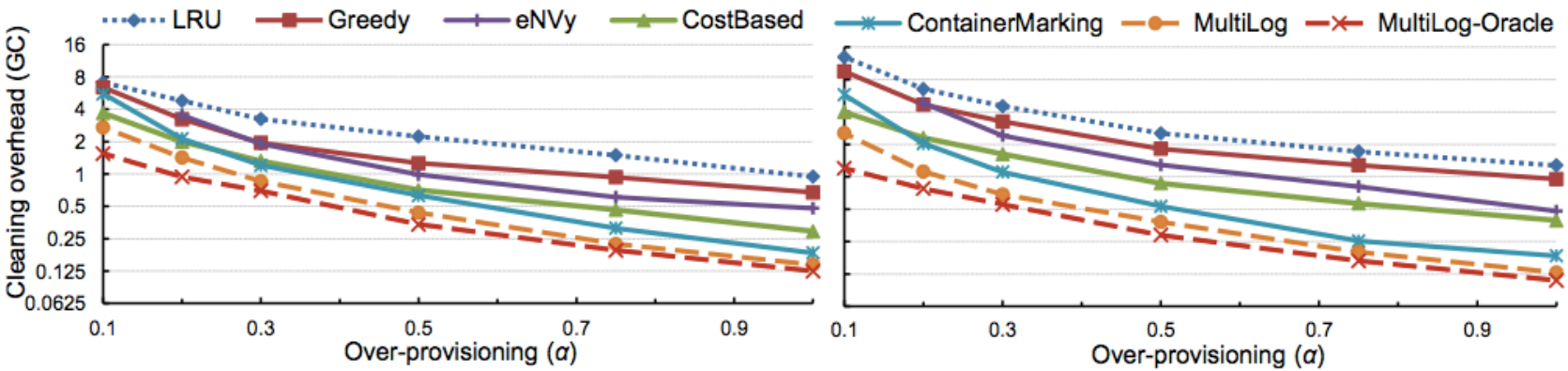


Figure 7: Cleaning cost as a function of over-provisioning for the TPC-C (a) and TATP (b) I/O traces.

Fig.7から引用